GENERALIZED FAULT TREE ANALYSIS FOR REACTOR SAFETY

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By

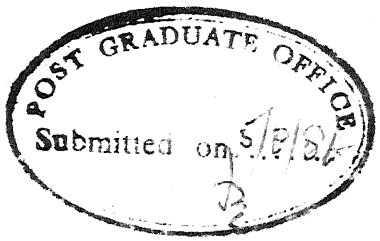
KAUSHALENDRA KUMAR SINGH

to the

NUCLEAR ENGINEERING AND TECHNOLOGY PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
AUGUST, 1986

TO

MY GRANDMOTHER



CERTIFICATE

This is to certify that the work embodied in the thesis "Generalized fault tree analysis for reactor safety", by Kaushalendra Kumar Singh, has been carried out under our supervision and has not been submitted elsewhere for a degree.

(K.Sri Ram)
Professor
Nuclear Engg. and Technology Programme
Indian Institute of Technology
Kanpur, India.

(Prabhat Munshi)
Lecturer
Nuclear Engg. and Technology Programme
Indian Institute of Technology
Kanpur, India.

August, 1986.

ACKNOWLEDGEMENTS

I am extremely grateful to my supervisors, Dr. K. Sri Ram and Mr. Prabhat Munshi for their excellent guidance and their constant encouragement and inspiration. They treated me with considerable affection and respect, and it was a pleasure to work under their guidance.

I also take this opportunity to thank all my friends, specially Satya, Anurag, Ajit, Nagendra and Rakesh, for their help.

Words can not express my sense of indebtness to my uncle Dr. Indra RajSingh, who is perennial source of inspiration and encouragement.

I would also like to express my gratitude to Shri G.L.Misra for his excellent typing of this thesis and Shri Gopal Krishna Tewari for his neat cyclostyling.

K.K.Singh

AUGUST, 1986. K.K. SINGH

C O N T E N T S

			Page
Chapter 1		INTRODUCTION	
	1.1	Introduction	1
Chapter 2		FAULT TREE CONSTRUCTION	
	2.1	Introduction	5
	2.2	Fault Trees	7
		2.2.1 Gate Symbols	7
		2.2.2 Event Symbols	16
	2.3	Finding Top Events	16
	2.4	Component Failure Characteristics	18
	2.5	Procedure for Fault tree Construction	18
Chapter 3		QUALITATIVE AND QUANTITATIVE ASPECTS OF SYSTEM ANALYSIS	
	3.1	Qualitative Aspects of System Analysis	25
		3.1.1 Cut Sets	25
		3.1.2 Path Sets	25
		3.1.3 Minimal Cut Sets and Minimal Path Sets	26
	3.2	Quantitative Aspects of System Analysis	
		3.2.1 Definitions	26
		3.2.2 Availability and Unavailability for Systems with Independent Basic Events	29
		3.2.2.1 Independent Basic Events	29
		3.2.2.2 System with one AND gate	29
		3.2.2.3 System with One OR gate	30
		3.2.3 Availability and Unavailability Calculation using Structure Functions	32
		3.2.3.1 Structure Functions	32

Page

3.2.3.2 System Representation in terms of Structure Functions 33

3.2.3.3 Unavailability Calculation using Structure Functions 34

3.2.4 Unavailability Calculations Using Minimal Cut Representation 36

Chapter 4 COMPUTER CODE AND RESULTS

4.1 Introduction 39

4.2 Features 39

4.3 Input Format 40

4.4 Constant Declaration 41

4.5 Computer Program

4.6 Results

4.7 Scope for Further Study

Chapter I

INTRODUCTION

1.1 Introduction :

Public concern about the safety of nuclear power stems from the fear of a potential release of large quantities of radioactivity from nuclear reactor in case of an accident and to some extent also about the possibility of slow release of small quantities of radioactivity from the spent fuel during its storage or from the high level immobilized waste resulting from fuel reprocessing.

In fact, all the radioactivity is produced in the Core of a reactor by the fission of heavy elements present in the fuel. The primary barrier to the release of this activity is provided by the cladding of the fuel. The fuel bundles in a pressurized heavy water reactor (PHWR) are kept in a pressure tube and cooled by pressurized heavy water and these pressure tubes are in turn fitted in a calandria filled with heavy water moderator.

If the clad of a fuel element is damaged the radioactivity is released to the coolant. However in the event of severe failure which causes damage to the pressure tubes, the radioactivity can enter the heavy water moderator.

The maximum credible accident namely LOSS OF COOLANT ACCIDENT (LOCA) can lead to meltdown of fuel bundles and can

lead to release of activity to the containment building. Efficacy of these multiple containment barriers is the subject of REACTOR ACCIDENT ANALYSIS. Basically the entire safety system is designed to ensure essentially zero release of radioactivity during the normal operation of the reactor and only a small release during highly improbable accident condition.

Analysis of accident conditions require data on integrity of all the components of a reactor. The chance of the release of radioactivity during an accident is the subject of SYSTEM RELIABILITY STUDIES. The Extensive risk assessment of nuclear power plants sponsored by the UNITED STATES ATOMIC ENERGY COMMISSION and completed in 1974, "WASH-1400 ,The Reactor Safety Study," [1] has been literally epochmaking. Professor N. Rasmussen analysed a vast spectrum of nuclear accidents, numerically ranked them in order of their probability of occurrence, and then assessed their potential consequence to the public. The Event tree, fault tree, and risk-consequence techniques used in this study are being widely adopted by the chemical and other industries. Rasmussen like studies are proliferating in Europe, Asia and the United States.

Rising public clamor regarding industrial hazards, coupled with strident consumerism and environmentalism, has had a profound impact on this decade. In Europe, following the serious industrial accidents at Flixborough, England and Cervesa, Italy [2] , there has been a rash of new

legislation requiring major risk studies prior to all new plant construction. In Britain, the new toxic substances Act could similarly affect every plant that has as much a single cylinder of compressed gas.

Even very recently, the world's worst nuclear disaster in the Kiev region (Chernobyl Mishap) of Soviet Union has revived the debate on the advisability of establishing more nuclear generation plants. Chernobyl is one of the major centers of the Soviet nuclear power programme. The four 1000 MW reactors there, are based on the unusual, but well known RMBK design - a water cooled; graphite moderated reactor system. The fuel is a ceramic, uranium oxide, with a melting point in the region of 3000 °C. It has been fused and then vaporised by the barbeque of thousands of tonnes of blazing graphite (known as the moderator) forming the core of the reactor. No one in the world, so for as is known, has experience of fighting a nuclear fire of this ferocity.

As the Soviet Union, with some international assistance, struggled to bring the situation at chernobyl and the affected zone under control, there was little doubt that this was a major setback to the image of nuclear power on a world scale. But there was equally little doubt that harnessing nuclear energy for civilian purposes was a growing concern both in the Soviet Union and elsewhere. Nuclear power had suffered a rude, unexpected blow, but there was no alternative to going ahead

with it - even while re-examining with rigour the entire gamut of safety issues.

Thus safety and risk analysis of a system has become an important subject. Fault tree technique provides tool to this aspect. The present work is an effort towards developing a fault tree computer code to evaluate probabilistically, systems modelled with Boolean Algebra technique.

Chapter 2

FAULT TREE CONSTRUCTION

2.1. Introduction :

A major goal of a reliability and safety analysis is to reduce the probability of failure and the attending human, economic and environmental losses.

The human losses include :

i) Death

ii) Injury

iii) Sickness or disability.

The economic losses are, for example :

1) production or service shutdown

2) off-specification products or services

3) Loss of capital equipment.

Some typical environmental losses are :

1) Air and water pollution

2) Other degradations of the environment such as Odour, vibration and noise.

Losses occur when one or more basic failure events create a system hazard. The three types of basic failure events most commonly encountered are

1) Events related to human beings :

a) operator error

b) design error

c) maintenance error

2) Events related to hardware, for example :

a) leakage of toxic fluid from a valve

b) loss of lubrication in a motor

c) incorrect measurement by a sensor.

3) Events related to the environment such as :

a) Earthquakes or ground subsidence

b) Storm, flood, tornado

c) ignition caused by sparks or lightning.

System hazards are frequently caused by a combination of failure events i.e. hardware failures plus human error and/or environmental fault events. Some typical policies used to minimise hazards and risk include

i) equipment redundancies

ii) inspection and maintenance

iii) protective systems such as sprinklers, fire walls, relief valves and emergency cooling system

iv) alarm displays.

A primary purpose of a system hazards study is to identify the causal relationships between the basic human, hardware and environmental events which result in system failures and to find ways of ameliorating their impact by system redesign and upgrades.

The causal relations can be developed by fault trees, which are then analyzed both qualitatively and quantitatively.

After the combination of the basic failure events which lead to system hazards are identified, the system can be improved and the hazards reduced.

2.2 Fault trees :-

The structure of fault tree is shown in Fig. 2.1. The undesired events appeared as the top event and this is linked to more basic fault events by event statements and logic gates. The central advantage of the fault tree vis-a-vis other techniques such as 'Failure Modes and Effect Analysis' (FMEA) is that the analysis is restricted only to the identification of the system elements and events that lead to one particular undesired failure or accident.

In order to find and visualize causal relations by fault trees, we require building blocks to classify and connect a large number of events. There are two types of building blocks : gate symbols and event symbols.

2.2.1 Gate Symbols :-

Gate symbols connect events according to their causal relations. The symbols for the gates are listed in Table 2.1. A gate may have one or more input events but only one output events.

i) AND gate : The output events of AND gates occur if all input events occur.

ii) OR gate : The output event of OR gates happen if any one of the input events occurs. Example of AND and OR gates are shown in Fig. 2.2.

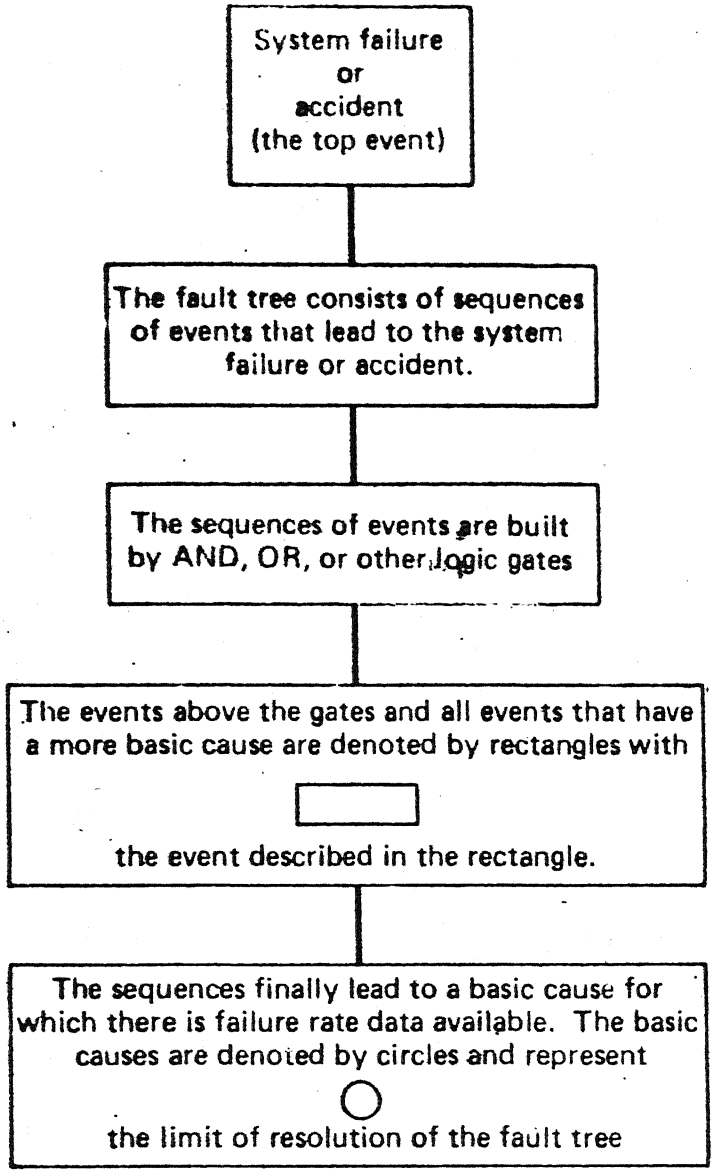
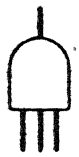
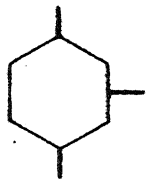
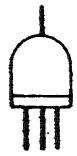
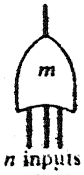


Figure 2.1. Fundamental structure of fault tree.

Ref [2]

TABLE 2.1 GATE SYMBOLS

	Gate Symbol	Gate Name	Causal Relation
1		AND gate	Output event occurs if all input events occur simultaneously.
2		OR gate	Output event occurs if any one of the input events occurs.
3		Inhibit gate	Input produces output when conditional event occurs.
4		Priority AND gate	Output event occurs if all input events occur in the order from left to right.
5		Exclusive OR gate	Output event occurs if one, but not both, of the input events occur.
6	 *n* inputs	*m* Out of *n* gate (voting or sample gate)	Output event occurs if *m* out of *n* input events occur.

Ref [2]

iii) INHIBIT gate : A hexagon, the inhibit gate is used to represent a probabilistic causal relation. The event at the bottom of the inhibit gate in Fig. 2.3 is called an input event, whereas the event to the side of the gate is a conditional event. The conditional event takes the form of an event conditioned by the input event. The output event occurs if both the input event and the conditional event occur. In other words the input event causes the output event with the (usually constant) probability of occurence of the conditional event. The inhibit gate frequently appears when an event occurs according to a demond. It is used primarily for convenience and can be replaced by an AND gate as shown in fig. 2.4.

iv) The Priority AND gate :- The priority AND gate is logically equivalent to an AND gate, with the additional requirement that the input event occur in a specific order namely from left to right. The representation equivalent to Fig.2.5 are shown in Fig.2.6 and 2.7.

v) Exclusive OR gate :-It describes a situation where the output event occurs if either one, not both, of the two input events occur. The equivalent representation is shown in Fig.2.8.

vi) An m-out-of-n voting gate :- It has n input events and the output event occurs if at least m out of n input events occur. Equivalent representation of Fig. 2.9 is shown in Fig. 2.10.

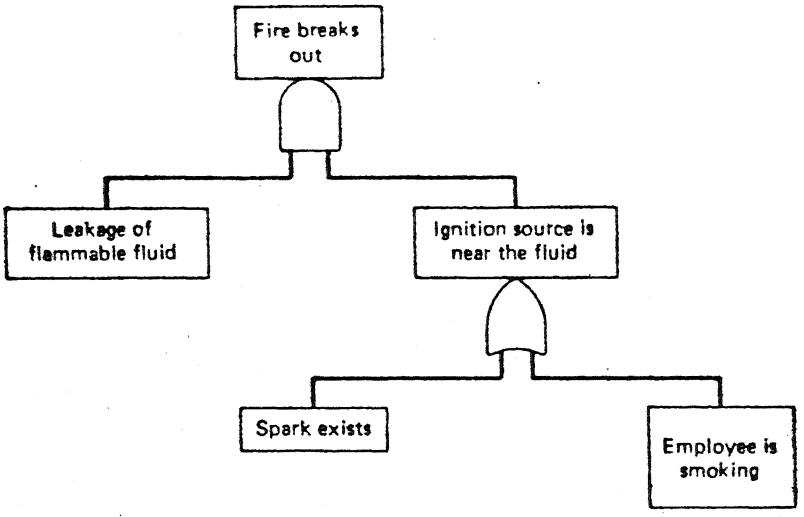


Figure 2.2. Example of AND gate and OR gate.

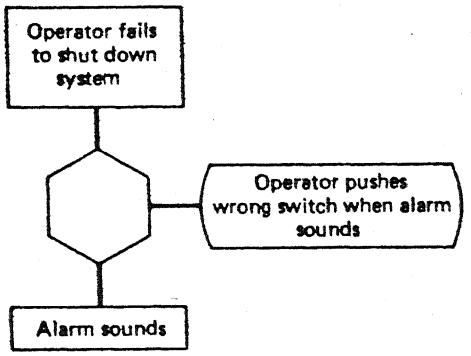


Figure 2.3. Example of inhibit gate.

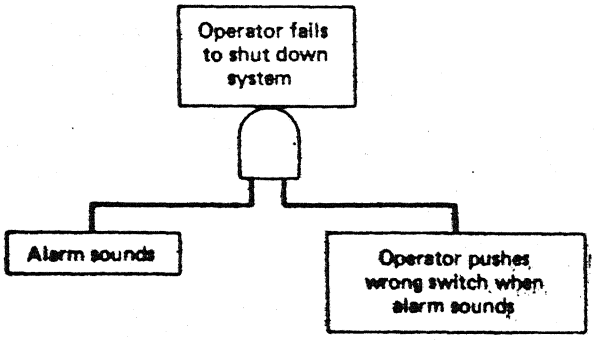


Figure 2.4. Equivalent expression to Fig. 2.3.

Ref [2]

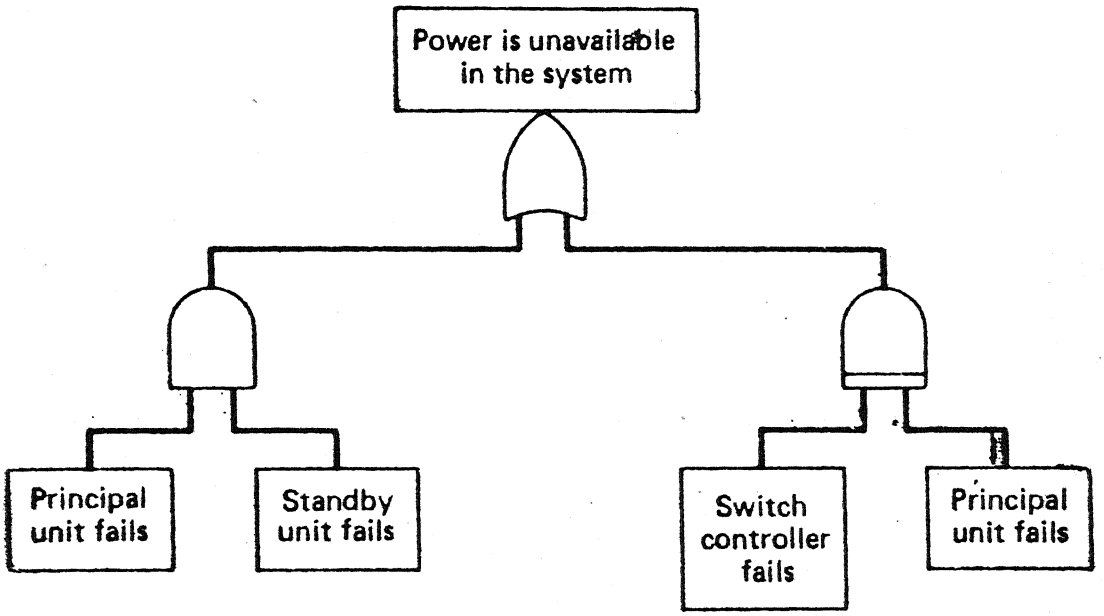


Figure 2.5. Example of priority AND gate,

Ref [2]

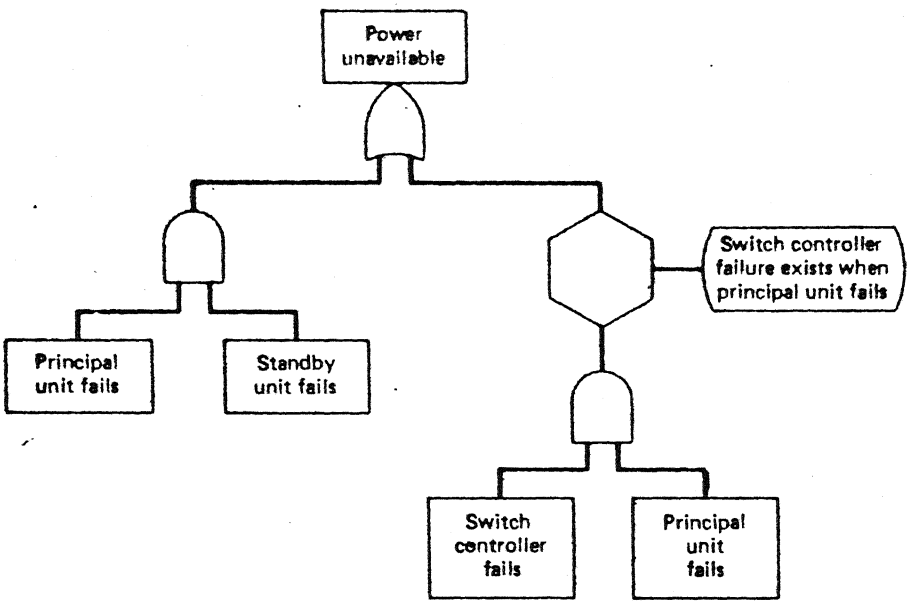


Figure 2.6. Equivalent expression to Fig. 2.5.

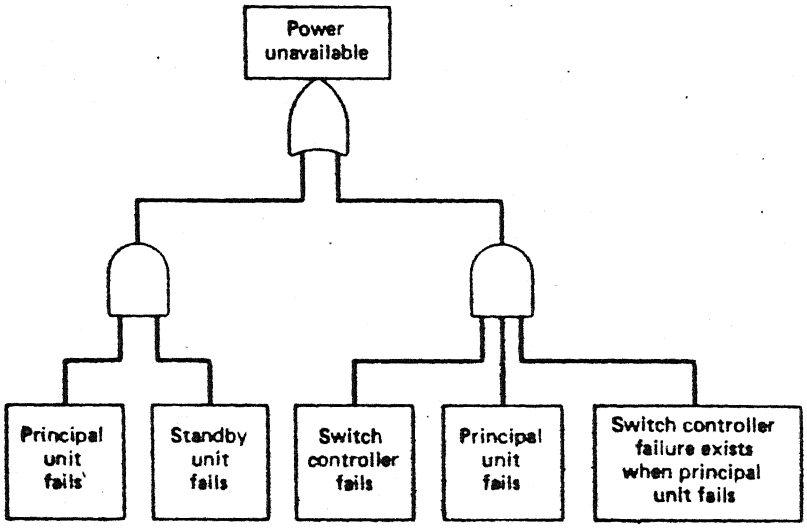


Figure 2.7. Equivalent expression to Fig. 2.5.

Ref [2]

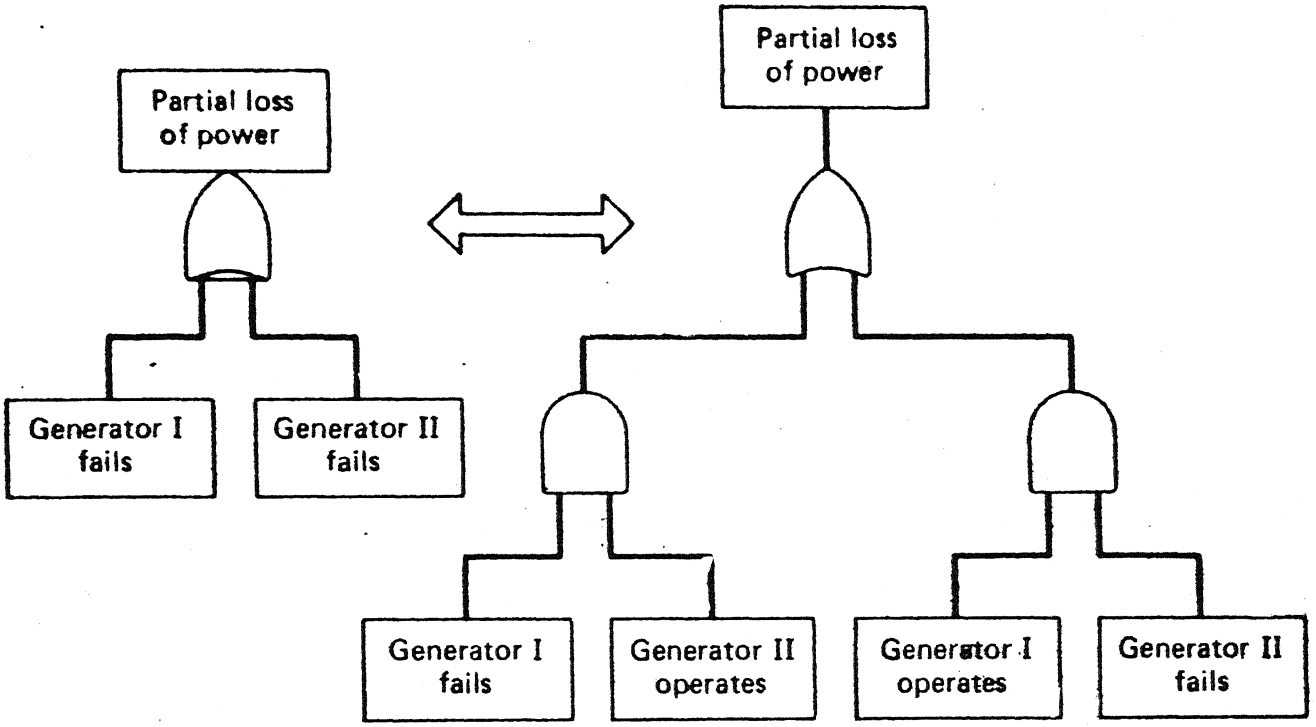


Figure 2.8. Example of exclusive OR gate and its equivalent expression.

Ref [2]

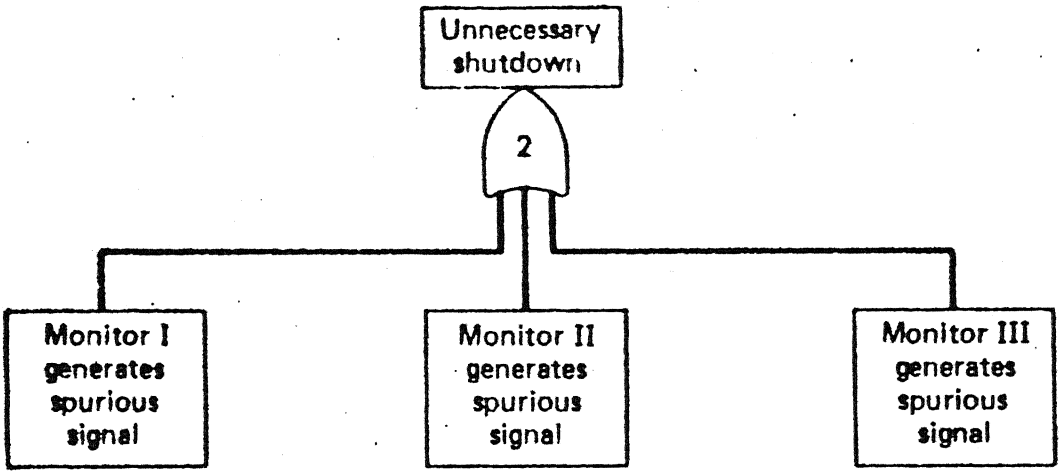


Figure 2.9. Example of two-out-of-three gate.

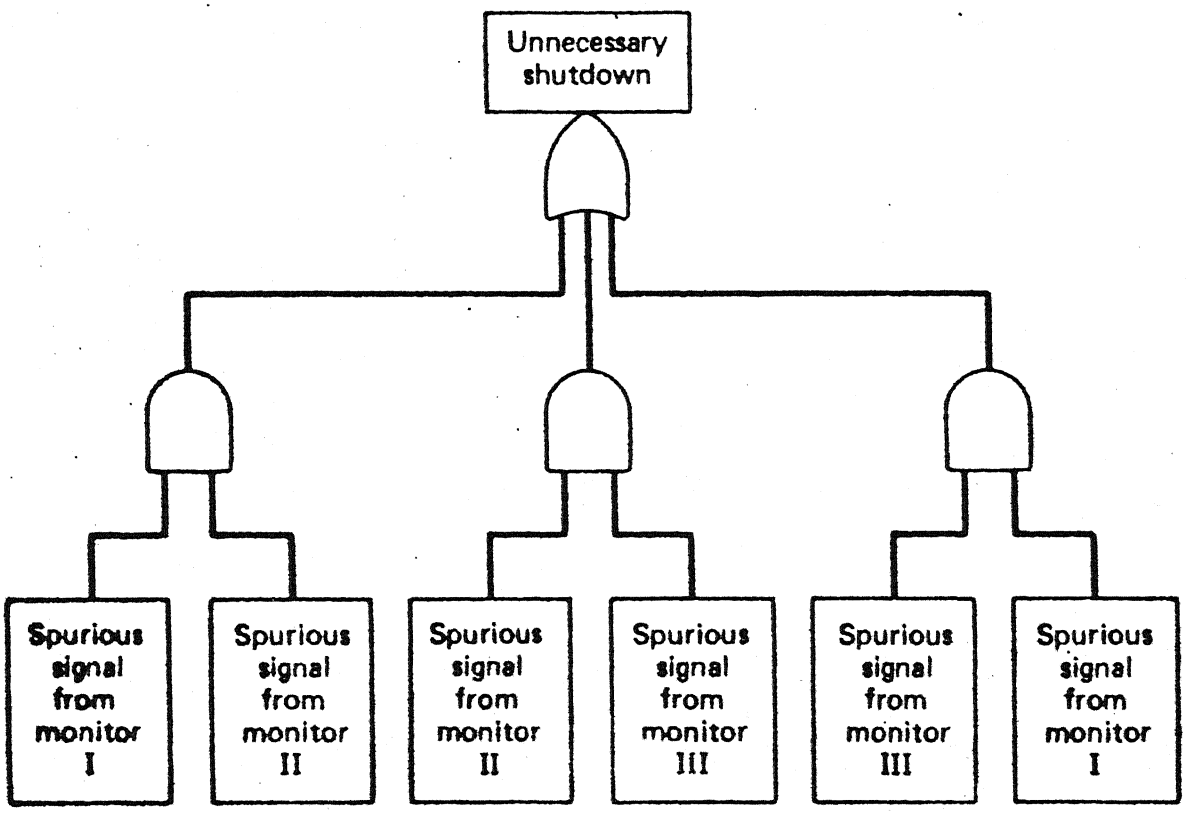


Figure 2.10. Equivalent expression to Fig. 2.9.

Ref [2]

2.2.2 Event Symbols :-

Event symbols are shown in Table 2.2. In the Schematic fault tree of Fig. 2.1, a rectangular box denotes a fault event resulting from a combination of more basic faults acting through logic gates. The circle designates a basic component failure (within the design envelope or environment) that represents the limit of resolution of a fault tree. In order to obtain a quantitative solution for a fault tree, circles must represent events for which reliability information is available. Events that appear as circles are called basic events.

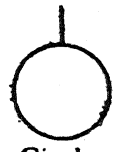
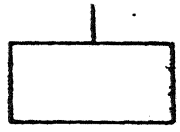
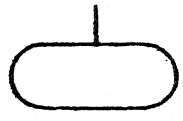
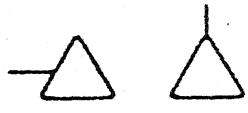
2.3 Finding Top Events :-

There are two approaches for analyzing causal relations : one is forward analysis, the other is backward analysis. A forward analysis starts with a set of failure events and proceeds forward, seeking possible consequences resulting from the events. A backward analysis begins with a system hazard and traces backward, searching for possible causes of the hazard.

The Event tree (ET), failure mode and effect analysis (FMEA), criticality analysis (CA), and preliminary hazards analysis (PHA), use the forward approach. Whereas Fault tree Analysis (FTA) uses backward approach.

The backward analysis i.e., the fault tree analysis is used to identify the causal relations leading to a given system hazard. The hazard itself becomes the top event of the fault tree.

TABLE 2.2 EVENT SYMBOLS

	Event Symbol	Meaning of Symbols
1	 Circle	Basic event with sufficient data
2	 Diamond	Undeveloped event
3	 Rectangle	Event represented by a gate
4	 Oval	Conditional event used with inhibit gate
5	 House	House event. Either occurring or not occurring
6	 Triangles	Transfer symbol

Ref [2]

2.4 Component Failure Characteristics :-

Component failures are fundamental in causal relation analysis. They are classified as either primary failures, secondary failures or command faults.

A primary failure is defined as a component being in the non-working state for which the component is held accountable, and repair action on the component is required to return the component to the working state. For example, "tank rupture due to metal fatigue" is a primary failure.

A secondary failure is the same as a primary failure except that the component is held accountable for the failure. Past or present excessive stresses placed on the component are responsible for the secondary failure. Examples are "fuse is opened by excessive current" and "Earthquake cracks storage tanks."

A Command fault is defined as the component being in the non-working state due to improper signals or noise and frequently repair action is not required to return the component to the working state. Examples are "switch randomly fails to open because of noise."

Fig. 2.11 describes the component failure characteristics.

2.5 Procedure For Fault tree Construction :-

A fault tree is a graphical representation of causal relations obtained when a system hazard is traced backward to

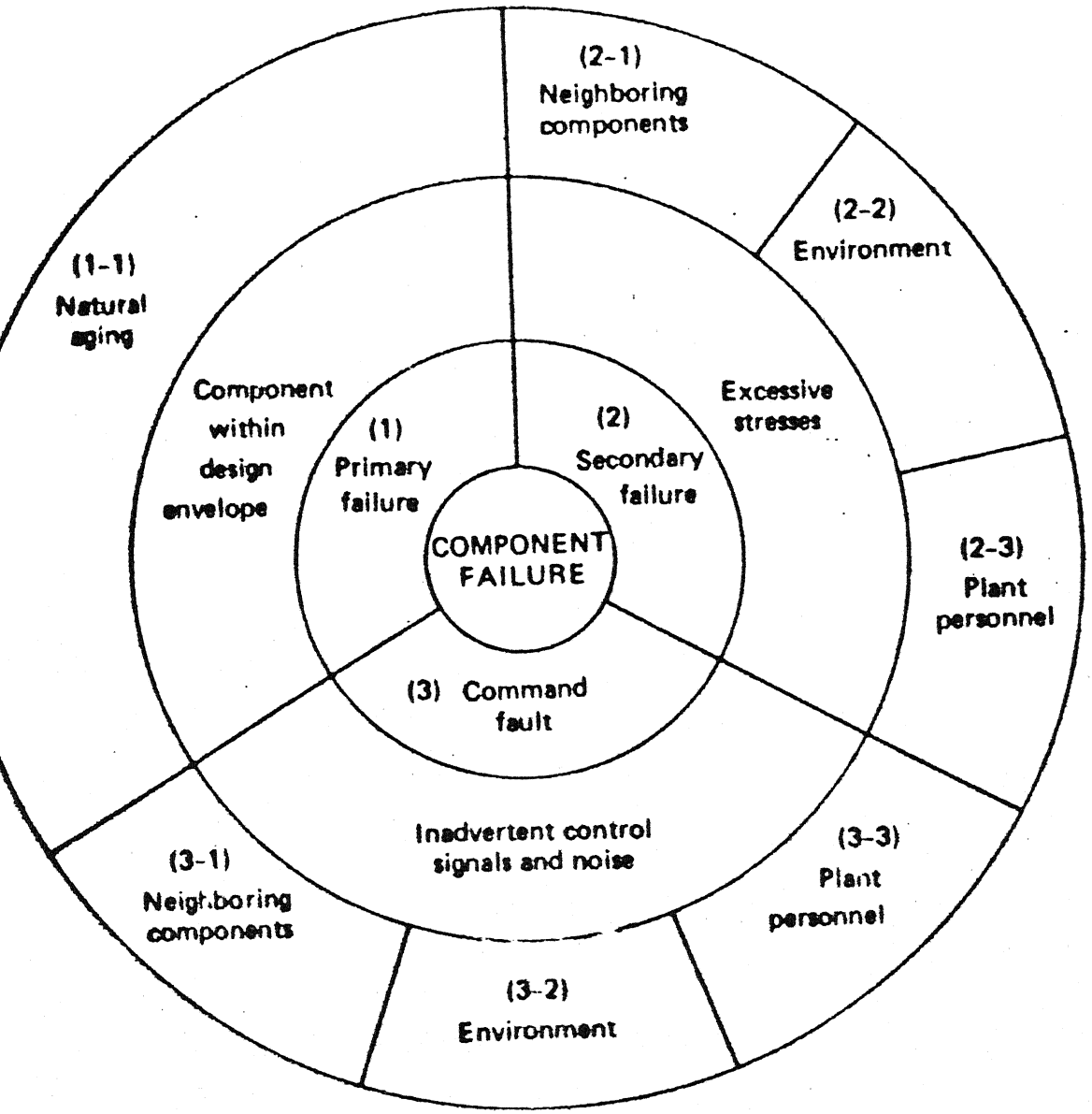


Figure 2.11. Component failure characteristics.

Ref [2]

search for its possible causes. The system hazard is then the top event of the fault tree.

Some heuristic guidelines are desirable for the construction of fault trees. These are summarized in Table 2.3 and Fig. 2.12. We have seven guidelines :

i) Replace an abstract event by a less abstract event.

ii) Classify an event into more elementary events.

iii) Identify distinct causes for an event.

iv) Couple trigger event with no protective action.

v) Find cooperative causes for an event.

vi) Pinpoint a component failure event.

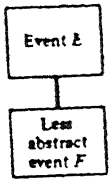
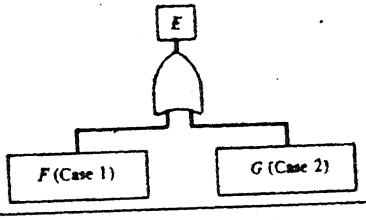
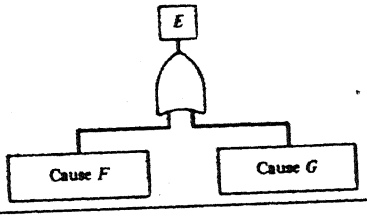
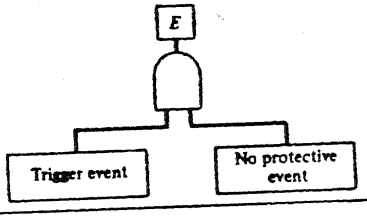
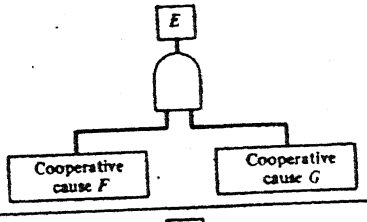
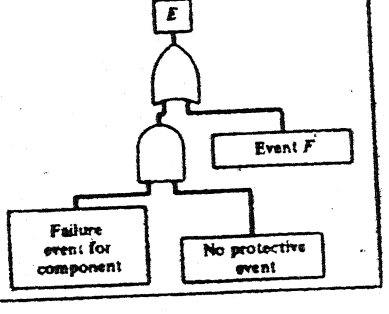
vii) Develop a component failure via Fig. 2.12.

Example 1 : As an example consider a pumping system shown in Fig. 2.13. The tank is filled in 10 min and empties in 50 min; thus, the cycle time is 1 hr. After the switch is closed, the timer is set to open the contacts in 10 min. If the mechanism fails then the alarm horn sounds and the operator opens the switch to prevent a tank rupture due to overfilling.

A fault tree with the top event of "tank rupture" is shown in Fig. 2.14.

A primary operator failure means that the operator functioning within the design envelope fails to push the panic button when alarm sounds. The secondary operator failure is, for example, "operator has been killed by a fire when the alarm sounded." The command fault for the operator is "no alarm sounds."

Table 2.3 Heuristic Guidelines for Fault Tree Construction

	Development Policy	Corresponding Part of Fault Tree
1	Equivalent but less abstract event F	
2	Classification of event E	
3	Distinct causes for event E	
4	Trigger versus no protective event	
5	Cooperative cause	
6	Pinpoint a component failure event	

Ref [2]

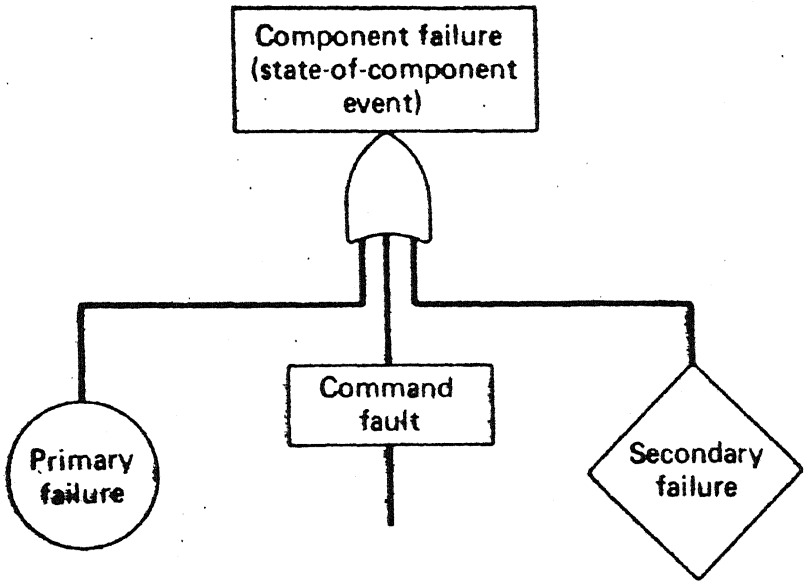


Figure 2.12. Development of a component failure (State-of-component event).

Ref [2]

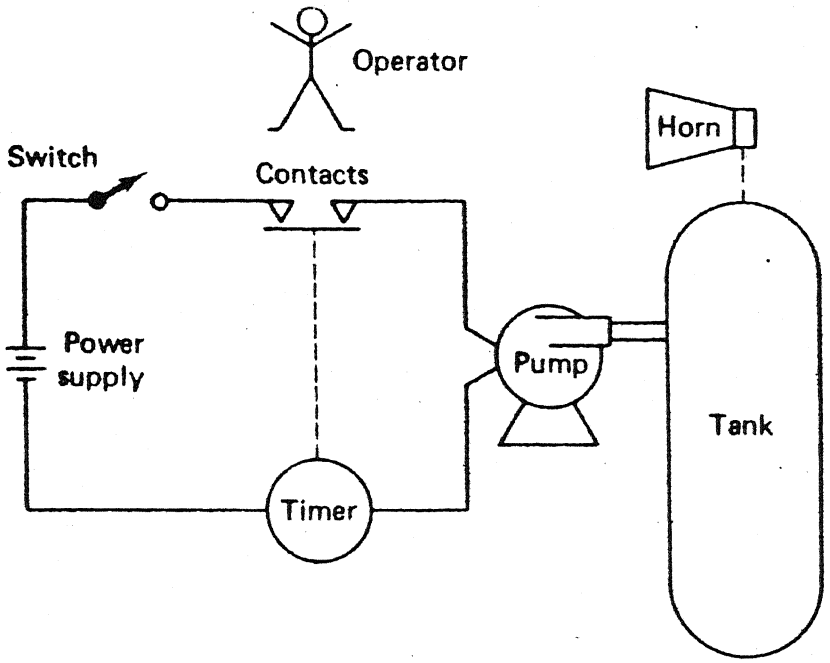


Figure 2.13. Schematic diagram for a pumping system.

Ref [2]

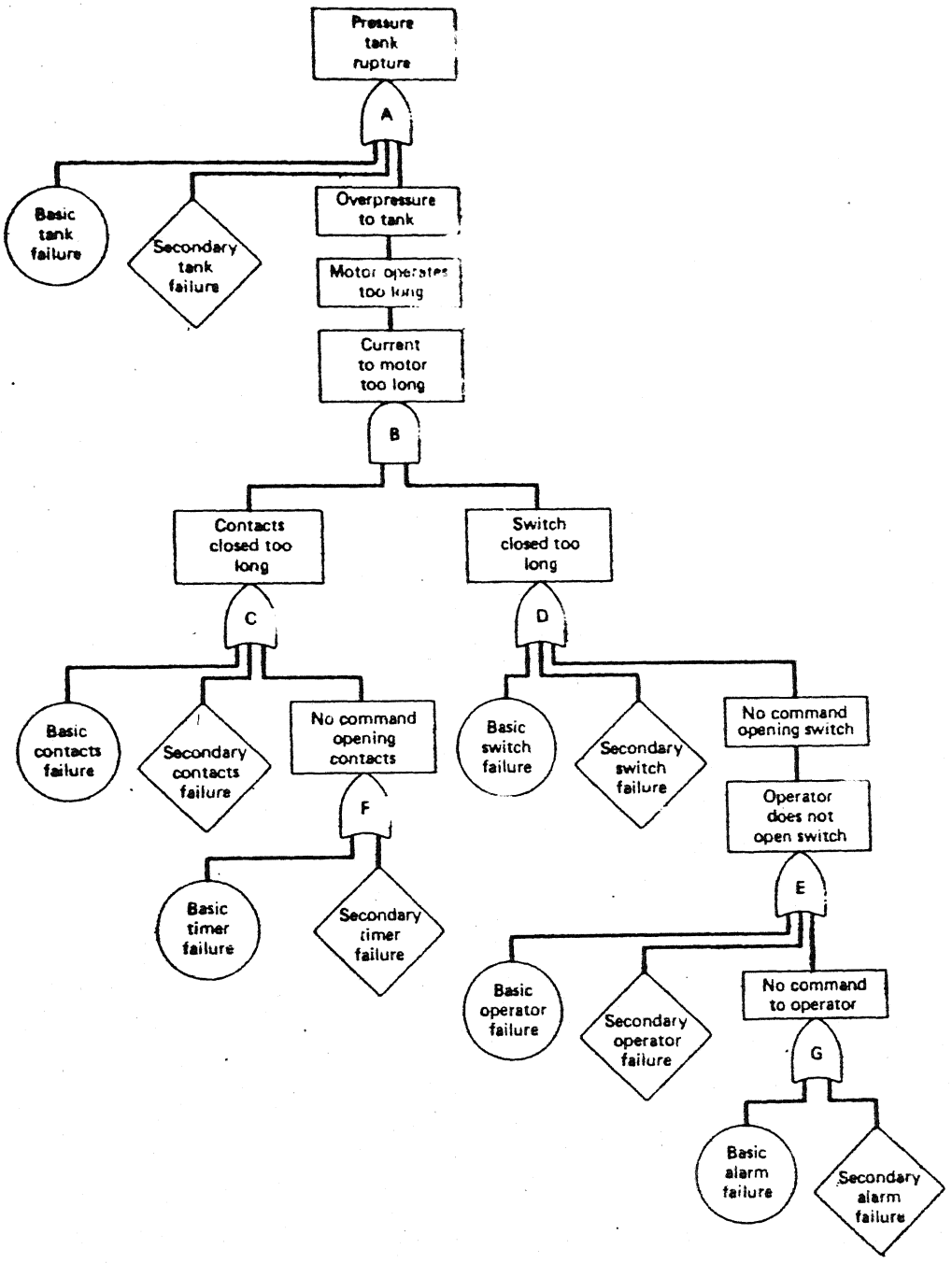


Figure 2.14. Fault tree for pumping system.

Ref [2]

Chapter 3

QUALITATIVE AND QUANTITATIVE ASPECT OF SYSTEM ANALYSIS

3.1 QUALITATIVE ASPECTS OF SYSTEM ANALYSIS

System failure can occur in many different ways. Each unique way is a system failure mode, and involves single or multiple component failures. To reduce the chance of a system failure, we must first identify the failure modes and then eliminate the most frequently occuring and/or highly probable ones. The fault tree method discussed in the previous chapter facilitatesthe discovery of failure modes.

3.1.1 Cut Sets :

A cut set is a collection of basic events; if all these basic events occur, the top event is guaranteed to occur.

3.1.2 Path Sets :

A path set is dual concept to the cut set. It is a collection of basic events and if none of the events in the set occur, the top event is guaranteed to not occur.

When the system has only one top event, the non-occurrence of the basic failure events in the path set ensures successful system operation. The non-occurrence does not guarantee system success when more than one top event is specified. In such cases a path set only ensures the non-occurrence of a particular top event.

3.1.3 Minimal Cut Sets and Minimal Path Sets :

A minimal cut set is such that if any basic event is removed from the set, the remaining events collectively are no longer a cut set. A cut set including some other sets is not a minimal cut set. The minimal cut set concept enables us to reduce the number of cut sets and the number of basic events involved in each cut set. This simplifies the analysis.

A minimal path set is a path set such that if any basic event is removed from the set, the remaining events collectively are no longer a path set.

3.2 QUANTITATIVE ASPECTS OF SYSTEM ANALYSIS

System success or failure can be described by a combination of top events defined by an OR combination of all system hazards into a composite fault tree (Fig. 3.1). In this chapter we shall discuss mainly the system availability and unavailability.

3.2.1 Definitions :

1) System availability A_s (t) : It is the probability that the top event does not exist at time t. This is the probability of the system's operating successfully when the top event refers to an OR combination of all system hazards. It is the probability of the nonoccurrence of a particular hazard when the top event is a single system hazard.

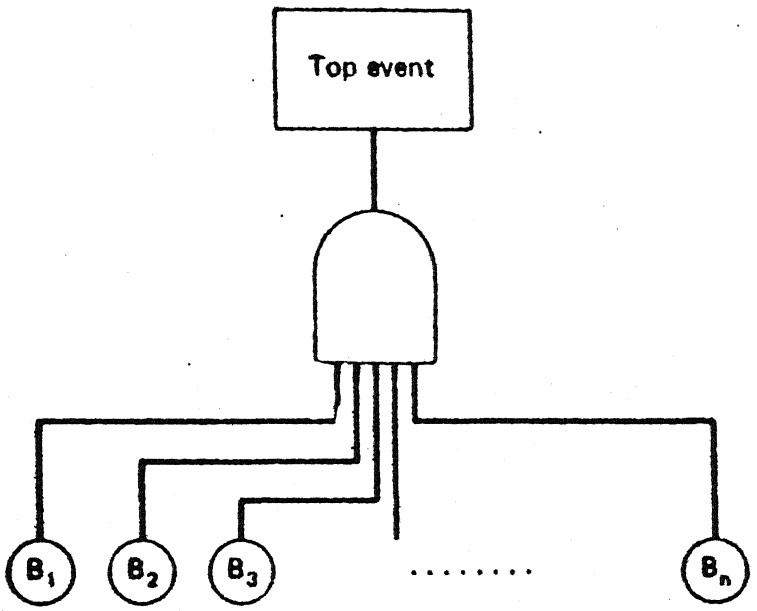


Figure 3.1. Gated AND fault tree.

Ref [2]

2) System Unavailability $Q_s(t)$:- It is the probability that the top event exists at time t. This is either the probability of system failure or the probability of a particular system hazard at time t, depending on the definition of the top event. The system unavailability is complementary to the availability, and the following identity holds

$$A_s(t) + Q_s(t) = 1 \qquad (3.1)$$

3) System Reliability $R_s(t)$:- Probability that the top event does not occur over the time interval $(0,t]$. The system reliability $R_s(t)$ requires continuation of the non-existence of the top event and differs from the system availability $A_s(t)$.

$$R_s(t) \leq A_s(t) \qquad (3.2)$$

4) System Unreliability $F_s(t)$:- Probability that the top event occurs before time t. This is the complement of the system reliability, and the identity

$$R_s(t) + F_s(t) = 1 \qquad (3.3)$$

holds. The system unreliability $F_s(t)$ is larger than or equal to the system unavailability.

$$F_s(t) \geq Q_s(t) \qquad (3.4)$$

3.2.2 Availability And Unavailability For Simple Systems With Independent Basic Events

3.2.2.1 Independent Basic Events :-

The usual assumption regarding basic events $B_1, \ldots, B_n$ is that they are independent, which means that occurrence of a givenbasic event is in no way affected by the occurrence of any other basic event. For independent basic events, the simultaneous existence probability $P_r(B_1 \cap B_2 \cap \ldots \cap B_n)$ reduces to

$$P_r(B_1 \cap B_2 \cap \ldots \cap B_n) = P_r(B_1)\ P_r(B_2) \ldots P_r(B_n) \qquad (3.5)$$

where the symbol $\cap$ represents the intersection of events $B_1, \ldots, B_n$.

3.2.2.1 System with One AND gate :-

Consider the fault tree of Fig. 3.1. Simultaneous existence of basic events $B_1, \ldots, B_n$ results in the top event. Thus the system unavailability $Q_s(t)$ is given by the probability that all basic events exist at time t :

$$Q_s(t) = P_r(B_1 \cap B_2 \cap \ldots \cap B_n) \qquad (3.6)$$

$$= P_r(B_1)\ P_r(B_2) \ldots P_r(B_n) \qquad (3.7)$$

For an AND gate with two input events, this reduces to

$$Q(t) = P_r(B_1)\ P_r(B_2) \qquad (3.8)$$

3.2.2.3 <u>System with one OR gate</u> :-

With reference to Fig. 3.2, the top event exists at time t if and only if at least one of the n basic events occurs at time t. Thus the system availability $A_s(t)$ and the system unavailability $Q_s(t)$ are given by

$$A_s(t) = P_r(\bar{B}_1 \cap \bar{B}_2 \cap \ldots \cap \bar{B}_n) \tag{3.9}$$

$$Q_s(t) = P_r(B_1 \cup B_2 \cup \ldots \cup B_n) \tag{3.10}$$

where the symbol $\cup$ denotes a union of the events, and $\bar{B}_i$ represents the complement of the event B_i; i.e. the event $\bar{B}_i$ means nonoccurrence of the event B_i at time t.

The independence of the basic event $B_1 \ldots B_n$ implies the independence of the complementary events $\bar{B}_1, \ldots, \bar{B}_n$. Thus $A_s(t)$ in (3.9) can be rewritten as

$$\begin{aligned} A_s(t) &= P_r(\bar{B}_1)\, P_r(\bar{B}_2) \ldots P_r(\bar{B}_n) \\ &= [1 - P_r(B_1)]\,[1 - P_r(B_2)] \ldots [1 - P_r(B_n)] \end{aligned} \tag{3.11}$$

The unavailability $Q_s(t)$ can be calculated by using (3.1)

$$\begin{aligned} Q_s(t) &= P_r(B_1 \cup B_2 \cup \ldots \cup B_n) \\ &= 1 - A_s(t) \\ &= 1 - [1 - P_r(B_1)]\,[1 - P_r(B_2)] \ldots [1 - P_r(B_n)] \end{aligned} \tag{3.12}$$

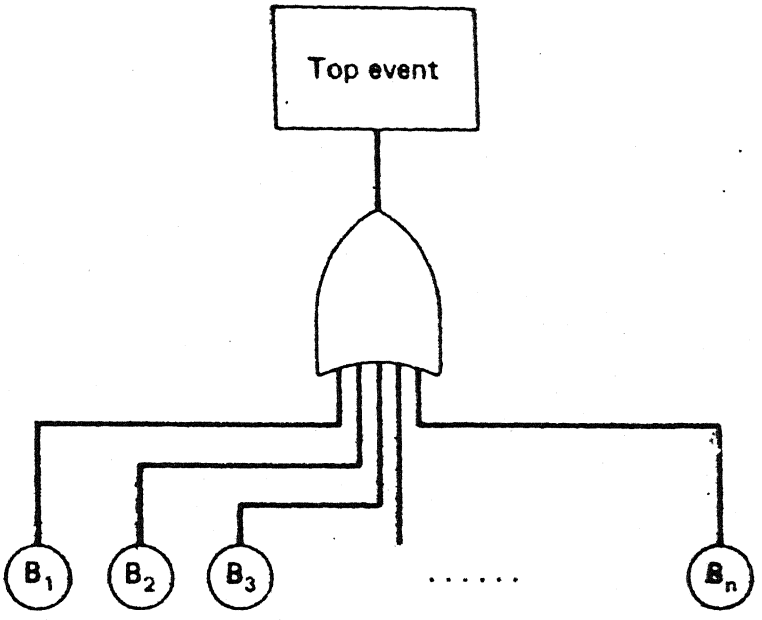


Figure 3.2. Gated OR fault tree.

Ref [2]

For n = 2, we have

$$Q_s(t) = P_r(B_1 \cup B_2)$$

$$= P_r(B_1) + P_r(B_2) - P_r(B_1)\, P_r(B_2) \qquad (3.13)$$

And for n = 3, we obtain

$$Q_s(t) = P_r(B_1 \cup B_2 \cup B_3)$$

$$= P_r(B_1) + P_r(B_2) + P_r(B_3)$$

$$- P_r(B_1)\, P_r(B_2) - P_r(B_2) P_r(B_3) - P_r(B_3) P_r(B_1)$$

$$+ P_r(B_1)\, P_r(B_2)\, P_r(B_3). \qquad (3.14)$$

3.2.3 AVAILABILITY AND UNAVAILABILITY CALCULATION USING STRUCTURE FUNCTIONS

3.2.3.1 Structure functions :-

It is possible to describe the state of the basic event or the system by a binary indicator variable. If we assign a binary indicator variable Y_i to the basic event i, then

$$Y_i = \begin{cases} 1, & \text{when the basic event is occurring} \\ 0, & \text{when the event is not occurring.} \end{cases}$$

Similarly, the top event is associated with a binary indicator variable $\Psi(Y)$ related to the state of the system by

$$\Psi(Y) = \begin{cases} 1, & \text{when the top event is occurring} \\ 0, & \text{when the top event is not occurring.} \end{cases}$$

Here $Y = (Y_1, Y_2, \ldots, Y_n)$ is the vector of basic event states. The function $\Psi(Y)$ is known as the structure function for the top event.

3.2.3.2 System Representation in Terms of Structure Function :-

The top event of the gated AND tree in Fig. 3.1 exists if and only if all basic events $B_1 \ldots B_n$ exist. In terms of the system structure function

$$\Psi(Y) = \Psi(Y_1, Y_2, \ldots, Y_n) = \bigwedge_{i=1}^{n} Y_i$$

$$= Y_1 \wedge Y_2 \wedge \ldots \wedge Y_n \qquad (3.15)$$

where Y_i is the indicator variable for the basic event B_i. The structure function can be expressed in terms of algebraic operators :

$$\Psi(Y) = \prod_{i=1}^{n} Y_i = Y_1, Y_2, \ldots, Y_n \qquad (3.16)$$

The gated OR tree of figure 3.2 fails (the top event exists) if any of the basic events $B_1, B_2, \ldots, B_n$ are occurring. The structure function is

$$\Psi(Y) = \bigvee_{i=1}^{n} Y_i = Y_1 \vee Y_2 \vee \ldots \vee Y_n \qquad (3.17)$$

and its algebraic form is

$$\Psi(Y) = 1 - \prod_{i=1}^{n} [1 - Y_i]$$

$$= 1 - [1 - Y_1][1 - Y_2] \ldots [1 - Y_n] \qquad (3.18)$$

The structure functions can be obtained in a stepwise manner for any fault tree. For example, the structure function for Fig. 3.3 is given as follows

$$\Psi_1(Y) = Y_B \wedge Y_C = Y_B Y_C$$

$$\Psi_2(Y) = Y_E \wedge Y_F = Y_E Y_F \qquad (3.19)$$

where

$\Psi_1(Y)$ is a structure function for the first AND gate

$\Psi_2(Y)$ is a structure function for the second AND gate.

Here, Y_B is an indicator variable for basic event B, etc. The structure function for the fault tree is

$$\begin{aligned}\Psi(Y) &= Y_A \vee \Psi_1(Y) \vee Y_D \vee \Psi_2(Y) \vee Y_G \\ &= 1 - [1-Y_A][1-\Psi_1(Y)][1-Y_D][1-\Psi_2(Y)][1-Y_G] \\ &= 1 - [1-Y_A][1-Y_B Y_C][1-Y_D][1-Y_E Y_F][1-Y_G]\end{aligned} \qquad (3.20)$$

3.2.3.3 <u>Unavailability Calculation Using Structure Functions</u> :

If we examine the system at some point in time, and the state of the basic event Y_i is assumed to be a Bernoulli random variable, then $\Psi(Y)$ is also a Bernoulli random variable. The probability of occurrencx of state $Y_i = 1$ is equal to the expected value of Y_i and to the probability of event B_i.

$$P_r(Y_i = 1) = P_r(B_i) = E(Y_i) \qquad (3.21)$$

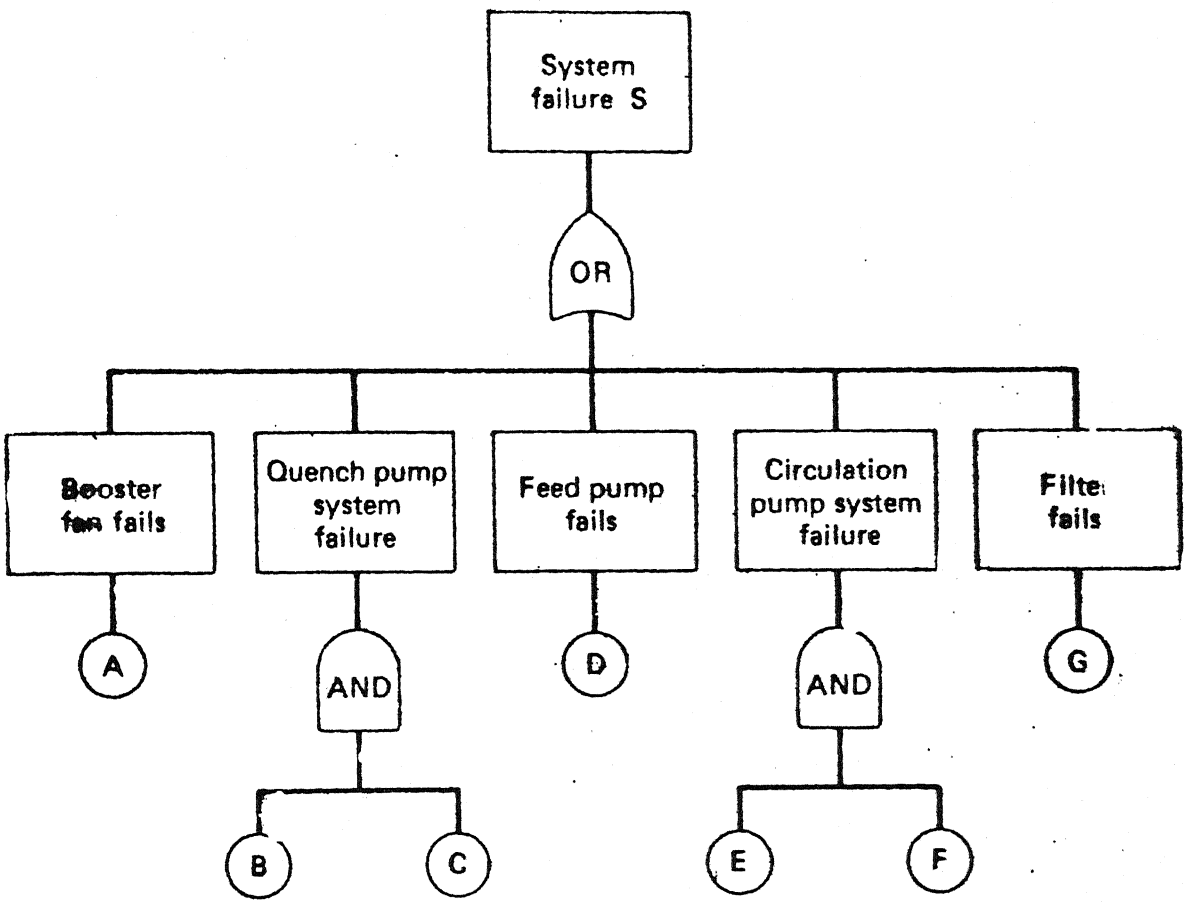


Figure 3.3. Fault tree for tail gas quench and clean up system.

Ref [2]

In fact, this probability is the unavailability $Q_i(t)$, or existence probability depending on whether basic event B_i is a component failure or human error or environmental impact. The probability of the top event i.e. the unavailability $Q_s(t)$, is the probability $P_r(\Psi(Y) = 1)$, or expectation $E(\Psi(Y))$. An alternative way of stating this is as follows

$$Q_s(t) = P_r(\text{top event}) = P_r(\Psi(Y) = 1) = E(\Psi(Y)) \qquad (3.22)$$

3.2.4 Unavailability Calculations Using Minimal Cut Representations

Consider a fault tree having the following m minimal cut sets

$\{B_{1,1}, B_{2,1}, \ldots, B_{n_1,1}\}$: cut sat 1

. . . . . .

$\{B_{1,j}, B_{2,j}, \ldots, B_{n_j,j}\}$: cut set j

. . . . . .

$\{B_{1,m}, B_{2,m}, \ldots, B_{n_m,m}\}$: cut set m

Denote by $Y_{i,j}$ the indicator variable for the event $B_{i,j}$. The top event occurs if and only if all basic events in a minimal cut set occur simultaneously. Thus the minimal cut set fault tree of Fig. 3.4 is equivalent to the fault tree. The structure function of this fault tree is

$$\Psi(Y) = \bigvee_{j=1}^{m} \left[\bigwedge_{i=1}^{n_j} Y_{i,j} \right] \qquad (3.23)$$

and its algebraic form is given by

$$\Psi(Y) = \bigvee_{j=1}^{m} \left[\prod_{i=1}^{n_j} Y_{i,j} \right]$$

$$= 1 - \prod_{j=1}^{m} \left[1 - \prod_{i=1}^{n_j} Y_{ij} \right] \qquad (3.24)$$

Let $K_j(Y)$ be a structure function for the AND gate G_j of Fig. 3.4 :

$$K_j(Y) = \prod_{i=1}^{n_j} Y_{i,j} \qquad (3.25)$$

The function $K_j(Y)$ is the jth minimal cut structure. Equation 3.24 can be written as

$$\Psi(Y) = 1 - \prod_{j=1}^{m} [1 - K_j(Y)] \qquad (3.26)$$

This equation gives a structure function of the fault tree in terms of minimal cut structures $K_j(Y)$'s. The structure function $\Psi(Y)$ can be expanded and simplified by the absorption law. System unavailability $Q_s(t)$ can be calculated by using (3.22).

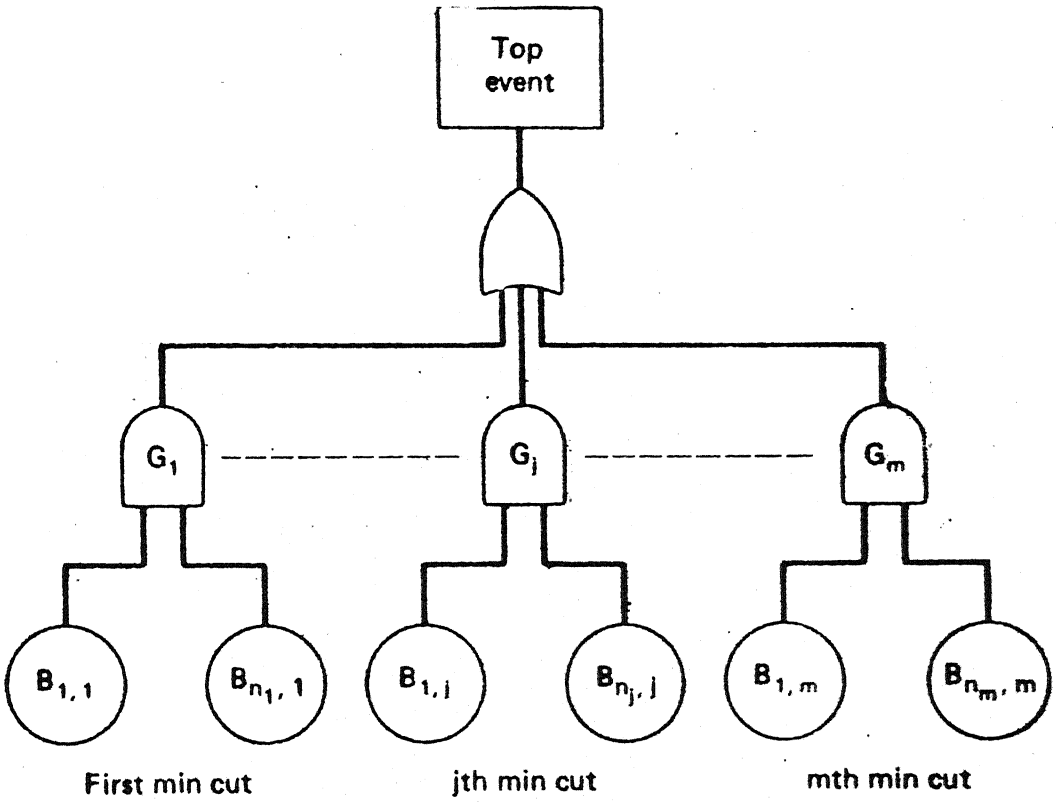


Figure 3.4. Minimal cut representation offault tree.

Ref [2]

Chapter 4

COMPUTER CODE AND RESULTS

4.1 Introduction :

A computer program in PASCAL is developed to evaluate the system reliability. The details of the computer program is discussed in this chapter.

4.2 Features :

(i) The developed computer program is completely general.

(ii) It determines the cut sets for the top event along with the intermediate events.

(iii) The input format is highly flexible. The whole fault tree is expressed in the form of simple boolean equations which relate the output of only one gate with its inputs.

(iv) Boolean equations are written only for AND and OR gates. If facult tree includes any other specific gates (such as Inhibit gate, m-out-of n gate), then these should be reduced to their equivalent using AND and OR gates as discussed in Section 2.2.1.

(v) There are two types of input events. One is basic and other is nonbasic. The Nonbasic can be further reduced to more basic events using gates.

(vi) The nonbasic events are called node. The first character of the node name is 'N'. The rest of the characters could be numerals or other characters except alphabet. For example N1, N15, N-12 are the valid node name, N1A2 is unvalid node name (3rd. character is alphabet).

(vii) The name of basic events could be any general name except nodename.

4.3 Input Format :

Three types of input lines are given in INPUT file.

(i) In the first type, each boolean equation is written in one line. The maximum characters in the line will be equal to STRINGLENGTH - a constant declared at the begining of program.

(a) AND gate is described as

N1 = (N2) . (N3) . (STEAMEXPL)

(b) OR gate is described as

N5 = (N6) + (STEAMEXPL) + (N5)

In both the above equations inputs to the gate are enclosed with the parenthesis. This is must. There is no restriction of any blanks inserted in the line.

(ii) The second type consists of those Nodenames for which the cutsets and probability evaluation is to be carried out. Only one nodename should be

inserted in one line.

For Example

N1

N5

N7

(iii) The third type, the probability values (of failures) are assigned to each of the basic inputs.

For Example

STEAMEXPL 1.0 E-5

Atleast one blank should be inserted between the basic and its corresponding probability values. In case only minimal cutsets are to be determined, these values may not be provided at all.

These three types of input lines could be given in any order in INPUT file

For Example

N1 = (N2) . (N3) . (STEAMEXPL) ---- FIRST LINE

N1 ---- SECOND LINE

N5 ---- THIRD LINE

STEAMEXPL 1.0 E-5 ---- FOURTH LINE

N5 = (N6) + (STEAMEXPL)+(N5) ---- FIFTH LINE

N7 ---- SIXETH LINE

4.4 Constant Declaration :

The following are the integer constants declared in the constant declaration of program. The values of these constant could be changed to suit the given problem.

(i) STRINGLENGTH : The constant STRINGLENGTH limits maximum character in one line or string.

(ii) VECTORLENGTH : provides the size of array of integers.

(iii) NOOFINPUTLINE : This constant limits the No of input lines present in the INPUT file. Moreover this constant should be more then the expected number of cutsets for the top event.

(iv) NOOFPROBNODE : This constant should have value more than the number of events (nodenames) for which cutsets are required.

4.5 Computer Program :

```
PROGRAM CUTSET(INPUT,OUTPUT);
  const
    STRINGLENGth = 60;
    VECTORLENGth = 20;
    NOOFINPUTLine = 100;
    NOOFPROBNOde = 5;
  type
    STRING = array [1..STRINGLENGth] of char;
    VECTOR = array [1..VECTORLENGth] of integer;
    MATRIX = array [1..NOOFINPUTLine, 1..STRINGLENGth]
      of char;
    ALPHABET = array [1..26] of char;
    ROW = array [1..3] of char;
    CUTMATRIX = array[1..NOOFPROBNOde, 1..NOOFINPUTLine,
      1..STRINGLENGth] of char;
  var
    CUTMATA : CUTMATRIX;
    MATA1,MATA2,MATA3,MATA4 : MATRIX;
    NODEMATA, PROBMATA : MATRIX;
    I,M1,M2,M3,M4,NONODE,NOPROB : integer;
  procedure  READSTRING( var S : STRING);
      var
        I : integer;
      begin
        I := 1;
        while not EOLN(INPUT) do
          begin
            READ(S[I]);
            I:= I+1;
          end;
        READLN;
        I := 0
      end;


  function  LENGTH( GIVENSTRING: STRING):integer;
      var
        I: integer;
      begin
        I:=1;
        while GIVENSTRING[I] # CHR(0) do
          I:=I+1;
        LENGTH := I-1;
        I := 0
      end;


  procedure CLEARSTRING( var GIVENSTRING: STRING);
      var
        I :integer;
      begin
        for I := 1 to STRINGLENGth do
          GIVENSTRING[I] := CHR(0);
        I := 0
      end;


  procedure CLEARVECTOr( var MATCHPOSN : VECTOR);
      var
```

```
    I : integer;
  begin
    for I := 1 to VECTORLENGth do
        MATCHPOSN[I] := 0;
    I := 0
  end;


procedure  COMPRESS(var INPUTSTRING : STRING);
  var
    OUTPUTSTRING : STRING;
    I,J,L : integer;
  begin
    CLEARSTRING(OUTPUTSTRING);
    L := LENGTH(INPUTSTRING);
    if (L # 0)  then
      begin
        J:= 0;
        for I := 1 to L do
          begin
            if (INPUTSTRING[I] # CHR(32)) then
              begin
                J := J+1;
                OUTPUTSTRING[J] := INPUTSTRING[I]
              end;
          end;
        CLEARSTRING(INPUTSTRING);
        for I := 1 to J do
          INPUTSTRING[I] := OUTPUTSTRING[I];
      end;
    CLEARSTRING(OUTPUTSTRING);
    I := 0; J := 0;
    L := 0
  end;


procedure  BRACKETREMove (var INPUTSTRING : STRING);
  var
    OUTPUTSTRING : STRING;
    I,J,L : integer;
  begin
    CLEARSTRING(OUTPUTSTRING);
    L:= LENGTH(INPUTSTRING);
    if (L#0) then
      begin
        J := 0;
        for I := 1 to L do
          begin
            if((INPUTSTRING[I] # ')') and (INPUTSTRING[I] #
            '(')) then
              begin
                J := J+1;
                OUTPUTSTRING[J] := INPUTSTRING[I]
              end;
          end;
        CLEARSTRING(INPUTSTRING);
        for I := 1 to J do
          INPUTSTRING[I] := OUTPUTSTRING[I];
      end;
    I := 0; J := 0; L := 0;
    CLEARSTRING(OUTPUTSTRING)
```

```
    end;

procedure COPY(GIVENSTRING: STRING; FROMPOSN,
                  NOCHAR: integer; var SUBSTRING : STRING);
    var
      I : integer;
    begin
      CLEARSTRING( SUBSTRING);
      for I := 1 to NOCHAR do
          SUBSTRING[I] := GIVENSTRING[FROMPOSN+I-1];
      I := 0
    end;


procedure  INSERT(var GIVENSTRING : STRING;
                  INSERTION : STRING; FROMPOSN: integer);
    var
      I : integer;
      GIVENLENGTH, INSERTLENGTH : integer;
    begin
      GIVENLENGTH := LENGTH(GIVENSTRING);
      INSERTLENGTH := LENGTH(INSERTION);
      for I:=(GIVENLENGTH+INSERTLENGTH) downto
      (FROMPOSN+INSERTLENGTH)  do
          GIVENSTRING[I] := GIVENSTRING[I-INSERTLENGTH];
      for I:= 1 to INSERTLENGTH do
          GIVENSTRING[FROMPOSN+I-1] :=INSERTION[I];
      I := 0; GIVENLENGTH := 0;
      INSERTLENGTH := 0
    end;


procedure  DELETE( var GIVENSTRING: STRING;
                  FROMPOSN,NOCHAR : integer);
    var
      I : integer;
    begin
      for I := FROMPOSN to (FROMPOSN + NOCHAR -1) do
          GIVENSTRING[I] := CHR(32);
      COMPRESS( GIVENSTRING);
      I := 0
    end;


procedure  RIGHTCONCATenate(var GIVENSTRING,TAILSTRING:
                                STRING);
    var
      L,F,I  : integer;
    begin
      F := LENGTH( TAILSTRING);
      L := LENGTH(GIVENSTRING);
      for I := (L+1) to (L+F) do
          GIVENSTRING[I] := TAILSTRING[(I-L)];
      L := 0; F := 0;
      I := 0
    end;


procedure  LEFTCONCATenate(var GIVENSTRING,FRONTSTRING:
```

```
                                    STRING);
    var
      L,F,I: integer;
    begin
      F:= LENGTH(FRONTSTRING);
      L:=LENGTH(GIVENSTRING);
      if (L # 0) then
        begin
          for I := (F+L) downto (F+1) do
              GIVENSTRING[I] := GIVENSTRING[I-F];
        end;

      for I:= 1 to F do
          GIVENSTRING[I] := FRONTSTRING[I];
      L := 0; F := 0;
      I := 0
    end;


procedure FINDCHAR(GIVENSTRING :STRING; SEARCHCHAR:
                   char; var MATCHPOSITION :VECTOR;
                   var NOOFMATCHES :integer);
    var
      L,J,I: integer;
    begin
      CLEARVECTOR (MATCHPOSITION);
      L := LENGTH(GIVENSTRING);
      J := 1;
      if (L # 0) then
        begin
          for I := 1 to L do
              if(GIVENSTRING[I] = SEARCHCHAR) then
                begin
                  MATCHPOSITION[J] := I;
                  J := J+1;
                end;
        end;
      NOOFMATCHES := J-1;
      L := 0; J := 0;
      I := 0
    end;

procedure CHECKBASIC( STRINGVALUE: STRING; var BASIC: boolean);
    var
      J,L,I : integer;
      TEMP : char;
      VALUE : ALPHABET;
    begin
      BASIC := false;
      L := LENGTH(STRINGVALUE);
      if (STRINGVALUE[1] # 'N') then
          BASIC := true
      else
        begin
          TEMP := 'a';
          for I := 1 to 26 do
            begin
              VALUE[I] := TEMP;
              TEMP := SUCC(TEMP)
            end;
```

```
            I := 2;
            while ((I<=L) and (not BASIC)) do
              begin
                J := 1;
                while ((J<=26) and (not BASIC)) do
                  begin
                    if (STRINGVALUe[I] = VALUE[J]).then
                        BASIC := true
                    else
                        J := J+1;
                  end;
                I := I+1
              end;
          end;
      end;

procedure COPYMAT(MATA:MATRIX;M: integer;var STRINGVALUe: STRING);
      var
        I : integer;
      begin
        I := 1;
        CLEARSTRING( STRINGVALUe);
        while (MATA[M,I] # CHR(0)) do
          begin
            STRINGVALUe[I] := MATA[M,I] ;
            I := I +1
          end;
        I := 0
      end;

procedure FORMMATRIX (STRINGVALUe : STRING; var M:integer;
                          var MATA : MATRIX);
      var
        I : integer;
      begin
        I := 1;
        while (STRINGVALUe[I] # CHR(0)) do
          begin
            MATA[M,I] := STRINGVALUe[I] ;
            I := I + 1
          end;
        I := 0
      end;

procedure NOTNODECHEck(var STRINGVALUe ,MODSTRING: STRING;
                           var NOTNODE : boolean);
      var
        L : integer;
      begin
        CLEARSTRING( MODSTRING);
        L := LENGTH(STRINGVALUe);
        if (STRINGVALUe[1] = '~') then
          begin
            NOTNODE := true;
            COPY (STRINGVALUe, 2, L-1, MODSTRING)
          end
        else
            NOTNODE := false;
```

```
      L := 0
    end;

procedure COMPARE(STRINGVALUe,COMPSTRING:STRING;var SAME:boolean);
    var
      I,L1,L2 : integer;
    begin
      L1 := LENGTH(STRINGVALUe);
      L2 := LENGTH(COMPSTRING);
      if (L1 = L2) then
        begin
          I :=1;   SAME := true;
          while((I<=L1) and SAME ) do
            begin
              if (STRINGVALUe[I] # COMPSTRING[I] ) then
                SAME := false
              else
                begin
                  SAME := true;
                  I := I +1
                end;
            end;
        end
      else
          SAME := false;
      I := 0; L1:= 0;
      L2 := 0
    end;

procedure CODEGENERAtion( STRINGVALUe : STRING;var CODE: ROW;
                          var MATA2: MATRIX; var M2: integer);
    var
      I,J,K,L : integer;
      SUBSTRING, COMPSTRING : STRING;
      TEMP : char;
      VALUE : ALPHABET;
      SAME : boolean;
      COUNTER : integer;
    begin
      CLEARSTRING( SUBSTRING);
      CLEARSTRING( COMPSTRING);
      TEMP := 'A';
      for i := 1 to 26 do
        begin
          VALUE[I] := TEMP;
          TEMP := SUCC(TEMP)
        end;
      L := LENGTH(STRINGVALUe);
      if( M2=0) then
        begin
          CODE[1] := VALUE[1];
          CODE[2] := VALUE[1];
          CODE[3] := VALUE[1];
          M2 := 1;
          for I := 1 to 3 do
              MATA2[ M2,I] := CODE[I];
          for I := 4 to 10 do
              MATA2[ M2,I] := CHR(32);
          for I := 1 to L do
```

```
            MATA2[ M2,I+10] := STRINGVALUE[I];
        end
      else
        begin
          SAME := false; COUNTER := 1;
          while ((COUNTER<= M2) and (not SAME)) do
            begin
              COPYMAT( MATA2, COUNTER, COMPSTRING);
              DELETE( COMPSTRING, 1, 10);
              COMPARE( STRINGVALUE, COMPSTRING, SAME);
              CLEARSTRING( COMPSTRING);
              COUNTER := COUNTER + 1
            end;
          if (SAME) then
            begin
              COUNTER := COUNTER -1;
              COPYMAT( MATA2, COUNTER, SUBSTRING);
              for I := 1 to 3 do
                  CODE[I] := SUBSTRING[I];
            end
          else
            begin
              M2 := COUNTER;
              COUNTER := COUNTER - 1;
              K := COUNTER mod 26;
              COUNTER := COUNTER div 26;
              J := COUNTER mod 26;
              I := COUNTER div 26;
              CODE[1] := VALUE[I+1];
              CODE[2] := VALUE[J+1];
              CODE[3] := VALUE[K+1];
              for I := 1 to 3 do
                  MATA2[ M2,I] := CODE[I];
              for I := 4 to 10 do
                  MATA2[ M2,I] := CHR(32);
              for I := 1 to L do
                  MATA2[ M2,I+10] := STRINGVALUE[I];
            end;
        end;
      CLEARSTRING( SUBSTRING);
      CLEARSTRING( COMPSTRING);
      COUNTER := 0; J := 0; K := 0; L := 0;
      TEMP := CHR(0);
      for I := 1 to 26 do
          VALUE[I] := CHR(0);
      I := 0
    end;

procedure INPUTMATA1 (var MATA1, NODEMATA, PROBMATA : MATRIX;
                      var M1, NONODE,NOPROB : integer);
    var
      INPUTSTRING : STRING;
      MATCHPOSN : VECTOR;
      I,J,L, NOMATCHES : integer;
      BASIC : boolean;
    begin
      CLEARSTRING(INPUTSTRING);
      CLEARVECTOR( MATCHPOSN);
      M1 := 0; NONODE := 0; NOPROB := 0;
      repeat
```

```
        READSTRING (INPUTSTRING);
        FINDCHAR( INPUTSTRING, '=', MATCHPOSN, NOMATCHES);
        if (NOMATCHES # 0) then
          begin
            COMPRESS (INPUTSTRING);
            M1 := M1 + 1;
            FORMMATRIX (INPUTSTRING, M1, MATA1);
            CLEARSTRING( INPUTSTRING)
          end
        else
          begin
            if (INPUTSTRING[1] # CHR(0)) then
              begin
                L := LENGTH( INPUTSTRING);
                I := 1;
                while((INPUTSTRING[I] = CHR(32)) and
                (I <= STRINGLENGth)) do
                    I := I + 1;
                I := 1 - 1;
                L := L - I;
                if( I # STRINGLENGth) then
                  begin
                    if( I # 0) then
                      begin
                        for J := 1 to L do
                            INPUTSTRING[J] := INPUTSTRING[I+J];
                        for J := L+I downto L+1 do
                            INPUTSTRING[J] := CHR(0);
                      end;
                    CHECKBASIC (INPUTSTRING, BASIC);
                    if (not BASIC) then
                      begin
                        NONODE := NONODE + 1;
                        FORMMATRIX(INPUTSTRING,NONODE,NODEMATA);
                        CLEARSTRING (INPUTSTRING)
                      end
                    else
                      begin
                        NOPROB :=NOPROB + 1;
                        FORMMATRIX(INPUTSTRING,NOPROB,PROBMATA);
                        CLEARSTRING (INPUTSTRING)
                      end;
                  end;
              end;
          end;
      until EOF( INPUT) ;
      CLEARSTRING( INPUTSTRING);
      CLEARVECTOR( MATCHPOSN);
      NOMATCHES := 0
    end;

procedure INPUTANALYZE(MATA1:MATRIX;M1 :integer;var MATA2, MATA3,
                       MATA4 :MATRIX; var M2, M3, M4 :integer);

    var
      I,J,L   : integer;
      CODE    : ROW;
      RIGHTPOSN,LEFTPOSN :VECTOR;
      FROMPOSN,COUNTER,NOCHAR : integer;
      NOMATCHES1,NOMATCHES2 : integer;
      MODSTRING , INPUTSTRING,SUBSTRING, CODE1  : STRING;
```

```
  IMPROBRAC, NOTNODE ,BASIC : boolean;
begin
  CLEARSTRING( MODSTRING);
  CLEARSTRING( INPUTSTRING);
  CLEARSTRING( SUBSTRING);
  CLEARSTRING( CODE1);
  CLEARVECTOr( RIGHTPOSN);
  CLEARVECTOr( LEFTPOSN);
  COUNTER := 1;
  IMPROBRAC := false;
  while((COUNTER <= M1) and (not IMPROBRAC)) do
    begin
      COPYMAT(MATA1, COUNTER, INPUTSTRING);
      FINDCHAR(INPUTSTRING, '(', LEFTPOSN,NOMATCHES1);
      FINDCHAR(INPUTSTRING, ')', RIGHTPOSN,NOMATCHES2);
      if (NOMATCHES1 = NOMATCHES2) then
        begin
          for I := 1 to NOMATCHES1 do
            begin
              FINDCHAR(INPUTSTRING, '(', LEFTPOSN,NOMATCHES1);
              FINDCHAR(INPUTSTRING, ')', RIGHTPOSN,NOMATCHES2);
              NOCHAR := RIGHTPOSN[I] - LEFTPOSN[I] - 1;
              FROMPOSN := LEFTPOSN[I] + 1;
              COPY (INPUTSTRING,FROMPOSN,NOCHAR,SUBSTRING);
              NOTNODECHEck(SUBSTRING, MODSTRING, NOTNODE);
              if (NOTNODE) then
                begin
                  FROMPOSN := FROMPOSN +1 ;
                  NOCHAR := NOCHAR - 1;
                  L := LENGTH(MODSTRING);
                  CLEARSTRING (SUBSTRING);
                  COPY(MODSTRING, 1, L, SUBSTRING)
                end;
              CHECKBASIC(SUBSTRING, BASIC);
              if (BASIC) then
                begin
                  CODEGENERAtion(SUBSTRING, CODE, MATA2, M2);
                  DELETE(INPUTSTRING, FROMPOSN, NOCHAR);
                  CLEARSTRING( CODE1);
                  for J := 1 to 3 do
                      CODE1[J] := CODE[J] ;
                  INSERT(INPUTSTRING, CODE1, FROMPOSN)
                end;
            end;
          M3 := COUNTER;
          FORMMATRIX(INPUTSTRING,M3, MATA3);
          M4 := COUNTER;
          BRACKETREMove( INPUTSTRING);
          FORMMATRIX( INPUTSTRING, M4, MATA4)
        end
      else
        begin
          IMPROBRAC := true;
          WRITE (' IMPROPER BRACKET IN INPUT LINE NO ');
          WRITELN (COUNTER)
        end;
      COUNTER := COUNTER +1
    end;
  CLEARSTRING( MODSTRING);
  CLEARSTRING( INPUTSTRING);
  CLEARSTRING( SUBSTRING);
```

```
      CLEARSTRING( CODE1);
      CLEARVECTOR( RIGHTPOSN);
      CLEARVECTOR( LEFTPOSN);
      for I := 1 to 3 do
          CODE1[I] := CHR(0);
      I := 0; J := 0; L := 0;
      FROMPOSN := 0; COUNTER := 0;
      NOCHAR := 0; NOMATCHES1 := 0;
      NOMATCHES2 := 0
    end;

procedure SEARCHNODE(MATA :MATRIX; NODENAME :STRING;
                     M :integer; var COUNTER : integer);
    var
      COMPSTRING, STRINGVALUE : STRING;
      MATCHPOSN : VECTOR;
      NOCHAR , NOMATCHES : integer;
      SAME : boolean;
    begin
      CLEARSTRING( COMPSTRING);
      CLEARVECTOR(MATCHPOSN);
      CLEARSTRING( STRINGVALUE);
      COUNTER := 1; SAME := false ;
      while ((COUNTER <= M) and (not SAME)) do
        begin
          COPYMAT(MATA ,COUNTER, STRINGVALUE);
          FINDCHAR(STRINGVALUE, '=', MATCHPOSN, NOMATCHES);
          NOCHAR := MATCHPOSN[1] - 1 ;
          COPY (STRINGVALUE, 1, NOCHAR, COMPSTRING);
          COMPARE(NODENAME, COMPSTRING, SAME);
          COUNTER := COUNTER + 1
        end;
      if (SAME) then
          COUNTER := COUNTER - 1
      else
        begin
          WRITELN (' NODENAME NOT FOUND IN INPUT ');
          COUNTER := 0
        end;
      CLEARSTRING( COMPSTRING);
      CLEARSTRING( STRINGVALUE);
      CLEARVECTOR( MATCHPOSN);
      NOMATCHES := 0;
      NOCHAR := 0
    end;

procedure CLEARMATRIX(var MATA: MATRIX; var M: integer);
    var
      I,J,L : integer;
      STRINGVALUE : STRING;
    begin
      CLEARSTRING( STRINGVALUE);
      for J := 1 to M do
        begin
          COPYMAT(MATA , J, STRINGVALUE);
          L := LENGTH(STRINGVALUE) ;
          CLEARSTRING (STRINGVALUE);
          for I := 1 to L do
              MATA [J,I] := CHR(0);
```

```
        end;
      M := 0;
      I := 0; J := 0; L := 0;

      CLEARSTRING( STRINGVALUE)
    end;


procedure CHECKANDOR(INPUTSTRING:STRING;var ANDED,ORED :boolean);
    var
      MATCHPOSN : VECTOR;
      NOMATCHES : integer;
    begin
      CLEARVECTOR( MATCHPOSN);
      FINDCHAR (INPUTSTRING, '.', MATCHPOSN, NOMATCHES);
      if (NOMATCHES # 0) then
        begin
          ANDED := true;
          ORED := false
        end
      else
        begin
          FINDCHAR(INPUTSTRING, '+', MATCHPOSN, NOMATCHES);
          if (NOMATCHES # 0 ) then
            begin
              ANDED := false;
              ORED := true
            end
          else
            begin
              ANDED := false;
              ORED := false
            end;
        end;
      CLEARVECTOR( MATCHPOSN);
      NOMATCHES := 0
    end;


procedure DOUBLENEGATion ( var STRINGVALUe : STRING);
    begin
      if( STRINGVALUE[1] = '~') then
        begin
          if (STRINGVALUE[2] = '~') then
            DELETE( STRINGVALUe , 1, 2) ;
        end;
    end;


procedure NODEBREAK (NODENAME,FIXEDSTRING,INPUTSTRING : STRING;
                     ANDED,ORED : boolean; var MATAB :MATRIX;
                     var M6 : integer);
    var
      I, NOMATCHES, NOCHAR : integer;
      STRINGVALUe : STRING;
      MATCHPOSN : VECTOR;

    begin
      CLEARSTRING( STRINGVALUe);
      CLEARVECTOR( MATCHPOSN);
      if (ANDED and (not ORED)) then
```

```
        begin
          LEFTCONCATenate (FIXEDSTRING,INPUTSTRING);
          M6 := M6 + 1;
          FORMMATRIX (FIXEDSTRING, M6, MATA6 )
        end
      else
        begin
          if (ORED and (not ANDED)) then
            begin
              FINDCHAR (INPUTSTRING, '+', MATCHPOSN, NOMATCHES);
              for I := 1 to (NOMATCHES + 1) do
                begin
                  FINDCHAR( INPUTSTRING, '+', MATCHPOSN, NOMATCHES);
                  if (NOMATCHES # 0) then
                    begin
                      NOCHAR := MATCHPOSN[1] - 1 ;
                      COPY( INPUTSTRING, 1, NOCHAR, STRINGVALUE);
                      LEFTCONCATenate( FIXEDSTRING , STRINGVALUE);
                      M6 := M6 + 1 ;
                      FORMMATRIX( FIXEDSTRING, M6, MATA6);
                      DELETE (FIXEDSTRING , 1 , NOCHAR ) ;
                      NOCHAR := MATCHPOSN[1] ;
                      DELETE ( INPUTSTRING, 1 , NOCHAR)
                    end
                  else
                    begin
                      LEFTCONCATenate ( FIXEDSTRING, INPUTSTRING);
                      M6 := M6 + 1 ;
                      FORMMATRIX (FIXEDSTRING, M6, MATA6)
                    end;
                end;
            end
          else
            begin
              LEFTCONCATenate (FIXEDSTRING, INPUTSTRING );
              M6 := M6 + 1 ;
              FORMMATRIX( FIXEDSTRING, M6, MATA6)
            end;
        end;
      I := 0; NOMATCHES := 0;
      NOCHAR := 0;
      CLEARVECTOR( MATCHPOSN);
      CLEARSTRING( STRINGVALUE)
    end;

procedure NOTNODEBREAK (NODENAME,FIXEDSTRING,INPUTSTRING : STRING;
                        ANDED,ORED : boolean; var MATA6 : MATRIX;
                        var M6 : integer ) ;
    var
      J,L,I,NOMATCHES : integer;
      STRINGVALUE : STRING;
      MATCHPOSN : VECTOR;
      NOCHAR : integer;
    begin
      CLEARSTRING( STRINGVALUE);
      CLEARVECTOR( MATCHPOSN);
      if(ANDED and (not ORED)) then
        begin
          FINDCHAR (INPUTSTRING, '.', MATCHPOSN, NOMATCHES);
          for I := 1 to (NOMATCHES +1) do
```

```
        begin
          FINDCHAR(INPUTSTRING, '.', MATCHPOSN, NOMATCHES);
          if (NOMATCHES # 0) then
            begin
              NOCHAR := MATCHPOSN[1] - 1 ;
              COPY( INPUTSTRING, 1, NOCHAR, STRINGVALUE);
              L := LENGTH( STRINGVALUE);
              for J:= (L+1) downto 2 do
                  STRINGVALUE[J] := STRINGVALUE[J-1];
              STRINGVALUE[1] := '-';
              DOUBLENEGATION( STRINGVALUE);
              LEFTCONCATENATE( FIXEDSTRING, STRINGVALUE);
              M6 := M6 + 1 ;
              FORMMATRIX( FIXEDSTRING, M6, MATA6);
              NOCHAR := LENGTH (STRINGVALUE);
              DELETE (FIXEDSTRING , 1 , NOCHAR );
              NOCHAR := MATCHPOSN[1] ;
              DELETE ( INPUTSTRING, 1 , NOCHAR)
            end
          else
            begin
              L := LENGTH( INPUTSTRING);
              for J := (L+1) downto 2 do
                  INPUTSTRING[J] := INPUTSTRING[J-1];
              INPUTSTRING[1] := '-';
              DOUBLENEGATION( INPUTSTRING);
              LEFTCONCATENATE( FIXEDSTRING, INPUTSTRING) ;
              M6 := M6 + 1 ;
              FORMMATRIX ( FIXEDSTRING, M6, MATA6 )
            end;
        end;
    end
  else
      if( ORED and (not ANDED)) then
        begin
          FINDCHAR( INPUTSTRING, '+', MATCHPOSN, NOMATCHES);
          for I := 1 to (NOMATCHES +1 ) do
            begin
              FINDCHAR( INPUTSTRING, '+', MATCHPOSN, NOMATCHES);
              if (NOMATCHES # 0) then
                begin
                  NOCHAR := MATCHPOSN[1] - 1 ;
                  COPY (INPUTSTRING, 1, NOCHAR, STRINGVALUE);
                  L := LENGTH( STRINGVALUE);
                  for J := (L+1) downto 2 do
                      STRINGVALUE[J] := STRINGVALUE[J-1];
                  STRINGVALUE[1] := '-';
                  DOUBLENEGATION( STRINGVALUE);
                  LEFTCONCATENATE( FIXEDSTRING, STRINGVALUE);
                  L := LENGTH( FIXEDSTRING);
                  for J := (L+1) downto 2 do
                      FIXEDSTRING[J] := FIXEDSTRING[J-1];
                  FIXEDSTRING[1] := ',' ;
                  NOCHAR := MATCHPOSN[1];
                  DELETE ( INPUTSTRING, 1, NOCHAR)
                end
              else
                begin
                  L:= LENGTH( INPUTSTRING);
                  for J := (L+1) downto 2 do
                      INPUTSTRING[J] := INPUTSTRING[J-1];
```

```
                    INPUTSTRING[1] := '~' ;
                    DOUBLENEGATion( INPUTSTRING);
                    LEFTCONCATenate( FIXEDSTRING, INPUTSTRING)
                  end;
              end;
            M6 := M6 + 1 ;
            FORMMATRIX (FIXEDSTRING, M6, MATA6)
          end
        else
          begin
            L := LENGTH( INPUTSTRING);
            for J := (L+1) downto 2 do
              INPUTSTRING[J] := INPUTSTRING[J-1];
            INPUTSTRING[1] := '~';
            DOUBLENEGATion( INPUTSTRING);
            LEFTCONCATenate( FIXEDSTRING, INPUTSIRING);
            M6 := M6 + 1 ;
            FORMMATRIX( FIXEDSTRING, M6, MATA6)
          end;
      J := 0; L := 0; I := 0;
      NOMATCHES := 0;
      NOCHAR := 0;
      CLEARVECTOr( MATCHPOSN);

      CLEARSTRING( STRINGVALUe)
    end;


procedure FORMCUTMAT( MATA : MATRIX; M: integer;var CUTMATA :
                      CUTMATRIX; var NUM : integer);
    var
      I,J,L : integer;
      STRINGVALUe : STRING;
    begin
      CLEARSTRING( STRINGVALUe);
      for I := 1 to M do
        begin
          COPYMAT( MATA, I, STRINGVALUe);
          L := LENGTH( STRINGVALUe);
          for J := 1 to L do
            CUTMATA[NUM,I,J] := MATA[I,J];
        end;
      I := 0; J := 0; L := 0;
      CLEARSTRING( STRINGVALUe)
    end;


procedure COPYCUTMAT (CUTMATA : CUTMATRIX; NUM : integer;
                      var MATA : MATRIX; var M: integer);
    var
      I,J : integer;
    begin
      I := 1; J := 1;
      while (CUTMATA[NUM,I,J] # CHR(0)) do
        begin
          while (CUTMATA[NUM,I,J] # CHR(0) ) do
            begin
              MATA[I,J] := CUTMATA[NUM,I,J] ;
              J := J + 1
            end;
          J := 1; I := I + 1 ;
```

```
        end;
      I := I - 1 ;
      L := 0;
      J := 0
    end;


procedure FIXVARPART ( STRINGVALUE : STRING; var FIXEDSTRING,
                       NODENAME : STRING);
    var
      I,J,K,L, NOMATCHES : integer;
      FROMPOSN , NOCHAR : integer;
      MATCHPOSN : VECTOR ;
      NOTNODE , BASIC : boolean;
      MODSTRING , TEMPSTRING : STRING;
    begin
      CLEARSTRING( MODSTRING);
      CLEARSTRING( TEMPSTRING);
      CLEARVECTOR( MATCHPOSN);
      FINDCHAR (STRINGVALUE, '.', MATCHPOSN, NOMATCHES);
      if (NOMATCHES = 0) then
        begin
          NOTNODECHECK (STRINGVALUE, MODSTRING, NOTNODE);
          if (NOTNODE) then
              CHECKBASIC( MODSTRING, BASIC)
          else
              CHECKBASIC ( STRINGVALUE , BASIC);
          if ( not BASIC ) then
            begin
              CLEARSTRING( FIXEDSTRING);
              L := LENGTH( STRINGVALUE);
              COPY( STRINGVALUE, 1, L, NODENAME)
            end
          else
            begin
              CLEARSTRING( NODENAME);
              L := LENGTH(STRINGVALUE);
              COPY (STRINGVALUE, 1, L, FIXEDSTRING);
              for J := (L+1) downto 2 do
                  FIXEDSTRING[J] := FIXEDSTRING[J-1];
              FIXEDSTRING[1] := '.'
            end;
        end
      else
        begin
          CLEARSTRING( NODENAME);
          CLEARSTRING( FIXEDSTRING);
          L := LENGTH( STRINGVALUE) ;
          STRINGVALUE[L+2] := '.' ;
          for I := (L+1) downto 2 do
              STRINGVALUE[I] := STRINGVALUE[I-1];
          STRINGVALUE[1] := '.' ;
          FINDCHAR ( STRINGVALUE, '.', MATCHPOSN, NOMATCHES);
          I := 1;
          while((NODENAME[1] = CHR(0)) and (I <= NOMATCHES -1)) do
            begin
              NOCHAR := MATCHPOSN[I+1] - MATCHPOSN[I] - 1 ;
              FROMPOSN := MATCHPOSN[I] + 1 ;
              COPY (STRINGVALUE, FROMPOSN, NOCHAR,TEMPSTRING);
              NOTNODECHECK( TEMPSTRING, MODSTRING, NOTNODE);
              if (NOTNODE) then
```

```
                CHECKBASIC (MODSTRING, BASIC)
            else
                CHECKBASIC( TEMPSTRING, BASIC);
            if (not BASIC) then
              begin
                L := LENGTH( TEMPSTRING);
                COPY ( TEMPSTRING, 1, L, NODENAME)
              end;
            I := I + 1
          end;
        I := I - 1 ;
        if ( NODENAME[1] # CHR(0)) then
          begin
            FROMPOSN := MATCHPOSN[I];
            NOCHAR := MATCHPOSN[I+1] - MATCHPOSN[I];
            DELETE ( STRINGVALUE, FROMPOSN, NOCHAR);
            L := LENGTH( STRINGVALUE);
            STRINGVALUE[L] := CHR(0);
            L := L - 1 ;
            COPY( STRINGVALUE, 1, L, FIXEDSTRING)
          end
        else
          begin
            L := LENGTH( STRINGVALUE);
            STRINGVALUE[L] := CHR(0);
            L := L + 1;
            COPY( STRINGVALUE, 1, L, FIXEDSTRING)
          end;
      end;
    I := 0; J := 0; K := 0;
    K := 0; NOMATCHES := 0;
    FROMPOSN := 0; NOCHAR := 0;
    CLEARVECTOR( MATCHPOSN);
    CLEARSTRING( MODSTRING);
    CLEARSTRING( TEMPSTRING)
  end;


procedure CUTSET( var CUTMATA: CUTMATRIX; var MATA1,MATA2,MATA3,
                  MATA4,NODEMATA,PROBMATA : MATRIX; var M1,M2,
                  M3,M4,NONODE,NOPROB : integer);
  var
    MODNAME,NODENAME,FIXEDSTRING,STRINGVALUE : STRING;
    MODSTRING,INPUTSTRING : STRING;
    COUNTER, I,J,K,L,M6,M5,M55  : integer;
    NOMATCHES, NOCHAR : integer;
    MATCHPOSN : VECTOR;
    BASIC, NOTNODE, ORED, ANDED : boolean;
    MATA5, MATA6 : MATRIX;
  begin
    CLEARSTRING( NODENAME);
    CLEARSTRING( MODNAME);
    CLEARSTRING( FIXEDSTRING);
    CLEARSTRING( STRINGVALUE);
    CLEARSTRING( MODSTRING);
    CLEARSTRING (INPUTSTRING);
    CLEARMATRIX( MATA5, M5);
    CLEARMATRIX( MATA6, M6);
    CLEARVECTOR( MATCHPOSN);
    INPUTMATA1 (MATA1, NODEMATA, PROBMATA,M1,NONODE,NOPROB);
    INPUTANALYZE ( MATA1, M1, MATA2,MATA3, MATA4, M2, M3,M4);
```

```
for K := 1 to NONODE do
  begin
    M5 := 1;
    COPYMAT( NODEMATA, K , NODENAME);
    FORMMATRIX( NODENAME, M5, MATA5);
    repeat
        COUNTER := 0; M55 := M5; M6 := 0 ;
        for I := 1 to M5 do
          begin
            COPYMAT( MATA5,I, STRINGVALUE);
            FIXVARPART( STRINGVALUE, FIXEDSTRING, NODENAME);
            if ( NODENAME[1] # CHR(0)) then
              begin
                NOTNODECHECK ( NODENAME,MODNAME, NOTNODE);
                if (NOTNODE ) then
                    SEARCHNODE( MATA4, MODNAME, M4, J)
                else
                    SEARCHNODE( MATA4, NODENAME, M4, J);
                COPYMAT (MATA4, J, INPUTSTRING);
                FINDCHAR( INPUTSTRING, '=', MATCHPOSN,
                          NOMATCHES);
                NOCHAR := MATCHPOSN[1];
                DELETE ( INPUTSTRING, 1, NOCHAR);
                CHECKANDOR( INPUTSTRING, ANDED, ORED);

                if (NOTNODE) then
                    NOTNODEBREAK( MODNAME,FIXEDSTRING,
                                  INPUTSTRING,
                                  ANDED, ORED, MATA6, M6)
                else
                    NODEBREAK ( NODENAME,FIXEDSTRING,
                                INPUTSTRING,
                                ANDED,ORED, MATA6, M6 );
              end
            else
              begin
                M6 := M6 + 1;
                FIXEDSTRING[1] := CHR(32);
                COMPRESS( FIXEDSTRING);
                FORMMATRIX( FIXEDSTRING, M6, MATA6);
                COUNTER := COUNTER + 1
              end;
          end;
        CLEARMATRIX ( MATA5, M5);
        for J := 1 to M6 do
          begin
            COPYMAT(MATA6, J, STRINGVALUE);
            FORMMATRIX ( STRINGVALUE, J, MATA5);
            CLEARSTRING ( STRINGVALUE)
          end;
        M5 := M6 ;
        CLEARMATRIX (MATA6,M6);
    until (COUNTER = M55) ;
    FORMOUTMAT( MATA5,M5, OUTMATA, K);
    CLEARMATRIX ( MATA5, M5)
  end;
I := 0; J := 0; K := 0;
L := 0;  M6 := 0; M5 := 0;
M55 := 0; COUNTER := 0;
NOMATCHES := 0; NOCHAR := 0;
CLEARVECTOR( MATCHPOSN);
```

```
        CLEARSTRING( NODENAME);
        CLEARSTRING( MODNAME);
        CLEARSTRING( FIXEDSTRING);
        CLEARSTRING( STRINGVALUE);
        CLEARSTRING( MODSTRING);
        CLEARSTRING( INPUTSTRING);
        CLEARMATRIX( MATA5, M5);
        CLEARMATRIX( MATA6, M6)
    end;

procedure WRITEMATRIX ( MATA : MATRIX; M : integer);
    var
      I,J : integer;
    begin
      if( M # 0) then
        begin
          for I := 1 to M do
            begin
              J := 1;
              WRITE(I:3, '      ');
              while( MATA[I,J] # CHR(0)) do
                begin
                  WRITE( MATA[I,J]);
                  J := J + 1
                end;
              WRITELN;
              WRITELN;
            end;
        end
      else
          WRITELN( 'Empty Lines');
      I := 0;
      J := 0
    end;

procedure WRITECUTMAtrix( CUTMATA: CUTMATRIX; NODEMATA :
                          MATRIX; NONODE : integer);
    var
      L,I,J,NUM : integer;
      NODENAME : STRING;
    begin
      CLEARSTRING (NODENAME);
      for NUM := 1 to NONODE do
        begin
          COPYMAT( NODEMATA, NUM, NODENAME);
          WRITE ('Cut Sets For Event-Node ');
          L := LENGTH( NODENAME);
          for  I := 1 to L do
              WRITE( NODENAME[I]);
          WRITELN;
          WRITELN( '---------------------------');
          WRITELN;
          I := 1; J := 1;
          while( CUTMATA[NUM,I,J] # CHR(0)) do
            begin
              WRITE(I:3, '      ');
              while( CUTMATA[NUM,I,J] # CHR(0)) do
                begin
                  WRITE( CUTMATA[NUM,I,J]);
```

```
                J := J + 1
              end;
            WRITELN;
            I := I + 1;
            J := 1
          end;
        CLEARSTRING( NODENAME);
        for I := 1 to 5 do
          WRITELN;
      end;
    I := 0; J := 0;
    NUM := 0
  end;



begin(MAIN)
  CLEARMATRIX( MATA1,M1);
  CLEARMATRIX( MATA2,M2);
  CLEARMATRIX( MATA3,M3);
  CLEARMATRIX( MATA4,M4);
  CLEARMATRIX( NODEMATA, NONODE);
  CLEARMATRIX( PROBMATA, NOPROB);
  CUTSET (CUTMATA,MATA1,MATA2,MATA3,MATA4,NODEMATA,
          PROBMATA, M1,M2,M3,M4,NONODE,NOPROB );
  WRITELN(  '      EXAMPLE PROBLEM - FAULT TREE ANALYSIS');
  WRITELN;
  WRITELN(  '     ***************************************');
  WRITELN;
  WRITELN;
  WRITE( 'Input Fault Tree Description in the Form ');
  WRITELN( 'of Boolean Equations ');
  WRITE( '-----------------------------------------');
  WRITELN( '----------------------');
  WRITELN;
  WRITEMATRIX( MATA1, M1);
  for I := 1 to 10 do
    WRITELN;
  WRITE( 'The Non-Basic Events For Which Cutsets & ');
  WRITELN( 'Reliabilities are Required ');
  WRITE( '-----------------------------------------');
  WRITELN( '--------------------------');
  WRITELN;
  WRITEMATRIX( NODEMATA, NONODE);
  for I := 1 to 10 do
    WRITELN;
  if ( NOPROB # 0) then
    begin
      WRITELN( 'Component Input Unavailabilities ');
      WRITELN( '--------------------------------');
      WRITELN;
      WRITEMATRIX( PROBMATA, NOPROB);
      for I := 1 to 10 do
        WRITELN;
    end;
  WRITELN( 'The Basic Fault Events & its Assigned Code');
  WRITELN( '------------------------------------------');
  WRITELN;
  WRITEMATRIX( MATA2, M2);
  for I := 1 to 10 do
```

```
    WRITELN;
  WRITEOUTMatrix( CHIMATA, NODEMATA, NONODE);
end.
```

4.6 Results :

The figure 4.1 shows the fault tree for which the cutsets are determined using the above computer program. The INHIBIT gate present in the fault tree is reduced to its equivalent as per section 2.2.1 and then fault tree is expressed in terms of simple equations. The input and output files are given here.

Input-File

```
N1 = ( N2 ) + ( STEAMEXPL)
N2 = (N3) + (N4)+(N 5)
N3 = (N6) . (N7)
N4 = (N7) . ( N10)
N5 = (N10).(N6)
N6 = (DIESEL1) + (PUMPAF) + ( PUMPA T/M )
N7 = (N8 ) + (N9)
N8 = (DIESEL2) + (PUMPBF)
N9 = (PUMPBT/M) + ( PUMPAT/M)
N10 = (N11) + (N12)
N11 = (PUMPCF)+(N13)
N13 = (N14) . (SVMS-102F)
N14 = (MVMS-102F) + (DIESEL1)
N12 = (PUMPCT/M) . ( N15)
N15 = (PUMPAT/M) + (PUMPBT/M)
N1
N7
N10
STEAMEXPL          1.0E-5
DIESEL1            3.0E-2
PUMPAF             1.0E-2
PUMPAT/M           8.0E-3
DIESEL2            3.0E-2
PUMPBF             1.0E-2
PUMPBT/M           8.0E-3
PUMPCF             5.0E-3
SVMS-102F          1.0E-3
MVMS-102F          5.0E-3
PUMPCT/M           2.0E-2
```

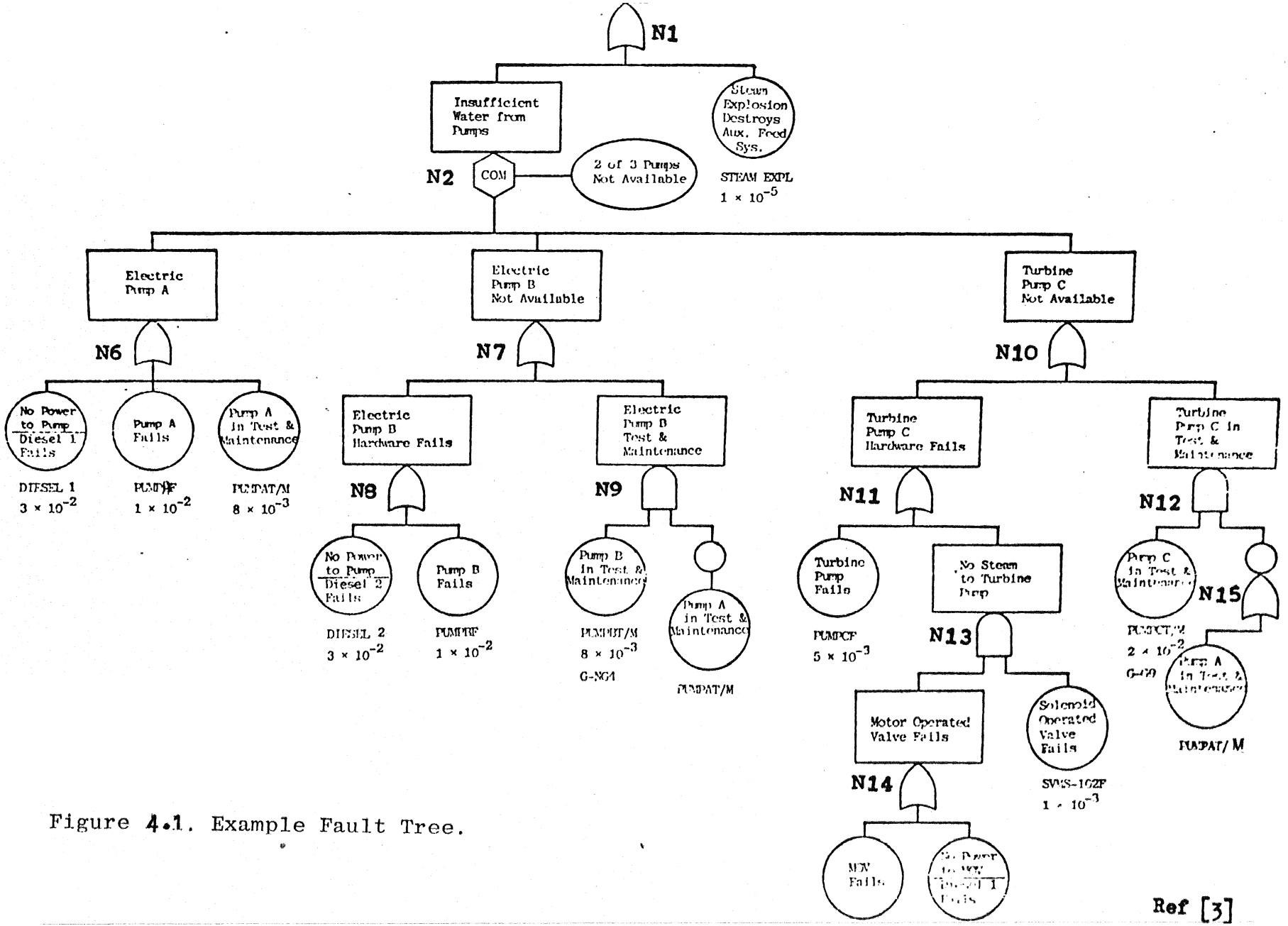


Figure 4.1. Example Fault Tree.

Ref [3]

Out-put

EXAMPLE PROBLEM - FAULT TREE ANALYSIS
****

Input Fault Tree Description in the Form of Boolean Equations

1. N1=(N2)+(STEAMEXPL)
2. N2=(N3)+(N4)+(N5)
3. N3=(N6).(N7)
4. N4=(N7).(N10)
5. N5=(N10).(N6)
6. N6=(DIESEL1)+(PUMPAF)+(PUMPAT/M)
7. N7=(N8)+(N9)
8. N8=(DIESEL2)+(PUMPBF)
9. N9=(PUMPBT/M)+(*PUMPAT/M)
10. N10=(N11)+(N12)
11. N11=(PUMPCF)+(N13)
12. N13=(N14).(SVMS-102F)
13. N14=(MVMS-102F)+(DIESEL1)
14. N12=(PUMPCT/M).(*N15)
15. N15=(PUMPAT/M)+(PUMPBT/M)

The Non-Basic Events For Which Cutsets & Reliabilities are Required

1. N1
2. N7
3. N10

Component Input Unavailabilities

1.	STEAMEXPL	1.0E-5
2.	DIESEL1	3.0E-2
3.	PUMPAF	1.0E-2
4.	PUMPAT/M	8.0E-3
5.	DIESEL2	3.0E-2
6.	PUMPBF	1.0E-2
7.	PUMPBT/M	8.0E-3
8.	PUMPCF	5.0E-3
9.	MVMS-102F	1.0E-3
10.	MVMS-102F	5.0E-3
11.	PUMPCT/M	2.0E-2

The Basic Fault Events & its Assigned Code

1.	AAA	STEAMEXPL
2.	AAB	DIESEL1
3.	AAC	PUMPAF
4.	AAD	PUMPAT/M
5.	AAE	DIESEL2
6.	AAF	PUMPBF
7.	AAG	PUMPBT/M

8.	AAH	PUMPCF
9.	AAI	SVMS-102F
10.	AAJ	MVMS-102F
11.	AAK	PUMPCT/M

Cut Sets For Event-Node N1

1. AAE.AAB
2. AAF.AAB
3. AAG.AAB
4. ~AAD.AAB
5. AAE.AAC
6. AAF.AAC
7. AAG.AAC
8. ~AAD.AAC
9. AAE.AAD
10. AAF.AAD
11. AAG.AAD
12. ~AAD.AAD
13. AAH.AAE
14. AAJ.AAI.AAE
15. AAB.AAI.AAE
16. ~AAG.~AAD.AAK.AAE
17. AAH.AAF
18. AAJ.AAI.AAF
19. AAB.AAI.AAF
20. ~AAG.~AAD.AAK.AAF
21. AAH.AAG
22. AAJ.AAI.AAG
23. AAB.AAI.AAG
24. ~AAG.~AAD.AAK.AAG
25. AAH.~AAD
26. AAJ.AAI.~AAD
27. AAB.AAI.~AAD
28. ~AAG.~AAD.AAK.~AAD
29. AAB.AAH
30. AAC.AAH
31. AAD.AAH
32. AAB.AAJ.AAI
33. AAC.AAJ.AAI
34. AAD.AAJ.AAI
35. AAB.AAB.AAI
36. AAC.AAB.AAI
37. AAD.AAB.AAI
38. AAB.~AAG.~AAD.AAK
39. AAC.~AAG.~AAD.AAK
40. AAD.~AAG.~AAD.AAK

41. AAA

Cut Sets For Event-Node N7

1. AAE
2. AAF
3. AAG
4. AAD

Cut Sets For Event-Node N10

1. AAH
2. AAJ.AAI
3. AAB.AAI
4. AAG.AAD.AAK

4.7 Scope for further study :

Normally the number of cutsets increases very fast with the increase in number of basic components. This requires large memory locations. Thus there is a need to write the very efficient program. Though care has been taken, still there is a scope to modify the present program.

Moreover the developed code determines only the cut sets. Thus there is a need to extend it to determine the minimal cutsets and reliability of the system.

BIBLIOGRAPHY

1. "Reactor Safety Study : An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants," Appendix III, WASH-1400, U.S. Nuclear Regulatory Commission (Oct. 1975)

2. Ernest J. Henly, Hiromitsu Kumamoto, "Reliability Engineering and Risk Assessment", (1981).

3. F.L. Leverenz, H. Kirch, "User's Guide For The WAM-BAM Computer Code", EPRI report, (Jan. 1976).

4. Tawfic A. Al-Kusayer, "Availability of the Emergency Core Cooling System of a Candu Pressurized Heavy-Water Reactor Following a Small Loss-OF-Coolant Accident", Nuclear Technology Vo. 69 (June 1985), p. 283-307.

5. H. Karwat, "Containments - Design Principles And Background", Nuclear Engineering and Design 90 (1985), p. 113-133.